

**МІНІСТЕРСТВО ФІНАНСІВ УКРАЇНИ
ДЕРЖАВНИЙ ПОДАТКОВИЙ УНІВЕРСИТЕТ**

**ФАКУЛЬТЕТ ФІНАНСІВ ТА ЦИФРОВИХ ТЕХНОЛОГІЙ
КАФЕДРА КОМП'ЮТЕРНИХ ТА ІНФОРМАЦІЙНИХ ТЕХНОЛОГІЙ І
СИСТЕМ**

КВАЛІФІКАЦІЙНА РОБОТА

**на тему: «Розробка відеогри для розвитку логіки мовою C# на платформі
Unity»**

Здобувача вищої освіти першого (бакалаврського) рівня,
IV курсу, групи КБ-21-1, спеціальності 122 «Комп'ютерні науки»,
освітня програма «Інформаційні управляючі системи і технології (за
галузями)»

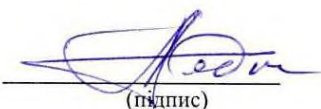


(підпис)

Пінчука Костянтина
Сергійовича

(прізвище, ім'я, по батькові)

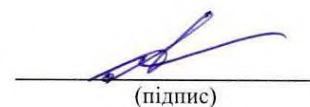
Науковий керівник



(підпис)

Редич Олександр
Володимирович к.е.н., доцент
(прізвище, ім'я, по батькові, науковий ступінь,
вчене звання)

Керівник освітньо-
професійної програми



(підпис)

Омельчук Антон
Анатолійович, к.е.н., доцент
(прізвище, ім'я, по батькові, науковий ступінь,
вчене звання)

Допущено до захисту за рішенням кафедри комп'ютерних та інформаційних
технологій і систем протокол від «13» травня 2025 № 18

Завідувач кафедри



(підпис)

Філоненко Михайло
Миколайович, к.ф.-м.н., доцент
(прізвище, ім'я, по батькові, науковий ступінь,
вчене звання)

Ірпінь – 2025

Реєстрація № 48

23.05.2025
(дата)



(підпис відповідальної особи)

Результати перевірки

до захисту
(до захисту, на доопрацювання)

23.05.2025
(дата)


(підпис)

Результат захисту

5 (відмінно)
(оцінка)

05.06.2025
(дата)


(підпис)

Голова комісії:


(підпис)

В.М. Франчук

Члени комісії:


(підпис)

М.М. Філоненко


(підпис)

С.М. Грищенко


(підпис)

А.А. Омельчук

БЛАНК ОЦІНЮВАННЯ КВАЛІФІКАЦІЙНОЇ РОБОТИ

Об'єкти контролю	Діапазон оцінки в балах	Фактична кількість балів	Примітки	Дата, П.І.Б.: керівника, голови комісії членів комісії, секретаря, (підписи)
ОЦІНЮВАННЯ КВАЛІФІКАЦІЙНОЇ РОБОТИ КОМІСІЄЮ ПО ЗАХИСТУ				
1. Зміст випускної роботи (якість теоретичного матеріалу, його зв'язок з практичним, вміння представляти матеріал в логічній послідовності, наявність висновків та пропозицій)	від 0 до 30	29		Редич О.В.  Франчук В.М. 
2. Дотримання вимог щодо оформлення	від 0 до 10	9		Філоненко М.М.  Грищенко С.М. 
3. Вчасне виконання роботи на всіх етапах у встановлені кафедрою терміни	від 0 до 5	5		Омельчук А.А.  Суховерхівка Л.П. 
4. Рецензент	від 0 до 5	4		
РАЗОМ БАЛІВ	від 0 до 50	47		

Оцінювання кваліфікаційної роботи науковим керівником – 44 балів.

ПЕРЕВЕДЕННЯ ОЦІНКИ ЗА 100-БАЛЬНОЮ ШКАЛОЮ ДО НАЦІОНАЛЬНОЇ ШКАЛИ ТА ШКАЛИ ЄКТС

Оцінка за 100-бальною шкалою	Оцінка за шкалою ЄКТС	Оцінка за національною шкалою	Дата	П.І.Б.: голови, членів комісії, секретаря, (підпис)
91	A	5 (відмінно)	05.06.2025	Франчук В.М.  Філоненко М.М.  Грищенко С.М.  Омельчук А.А.  Суховерхівка Л.П. 

ПРАВИЛА ПЕРЕВЕДЕННЯ ОЦІНОК ДО НАЦІОНАЛЬНОЇ ШКАЛИ ТА ШКАЛИ ЄКТС

Оцінка за 100-бальною шкалою	Оцінка за шкалою ЄКТС	Оцінка за національною шкалою
90-100	A	5 (відмінно)
80-89	B	4 (добре)
70-79	C	
60-69	D	
50-59	E	3 (задовільно)
35-49	FX	2 (незадовільно)

Факультет фінансів та цифрових технологій
Кафедра комп'ютерних та інформаційних технологій і систем
Галузь знань 12 «Інформаційні технології»
Спеціальність 122 «Комп'ютерні науки»
Освітня програма «Інформаційні управляючі системи і технології (за галузями)»

Затверджую

Завідувач кафедри  Філоненко М.М.
«15» листопада 2024

**Завдання на кваліфікаційну роботу
здобувачу вищої освіти**

Пінчуку Костянтину Сергійовичу

1. Тема кваліфікаційної роботи **«Розробка відеогри для розвитку логіки мовою C# на платформі Unity»**

Затверджена наказом ректора від «13» листопада 2024 № 1717

2. Строк здачі закінченої роботи «5» травня 2025

3. Мета роботи: створення та аналіз відеогри, що сприяє розвитку логічного мислення користувачів.

Об'єкт дослідження: відеоігрові середовища як засіб розвитку аналітичного мислення.

Предмет дослідження: інструментарій розробки двовимірної логічної відеогри на мові програмування C# в ігровому рушії Unity.

Орієнтовна база дослідження: Unity, приклади реалізації подібних проєктів.

4. Вхідні дані до роботи: матеріали офіційної документації Unity, наукові праці в галузі освітніх технологій та ігрової педагогіки.

5. Перелік графічного матеріалу: рисунки інтерфейсів додатку, структурні схеми, алгоритми, лістинги програм.

6. Зміст кваліфікаційної роботи (перелік питань за кожним розділом)

Розділ 1. Теоретико-методологічні основи розробки логічних відеоігор

1.1. Відеоігри як інструмент розвитку логічного мислення

1.2. Особливості побудови логічних ігор: жанри, принципи, підходи

1.3. Шахові механіки та фізичні моделі в іграх

1.4. Технології розробки логічних ігор: огляд та обґрунтування вибору C# і Unit

1.5. Аналіз подібних ігор та рішень

Розділ 2. Системно-аналітичне обґрунтування архітектури гри

2.1. Моделювання ігрового процесу: логіка, цілі, рівні складності

2.2. Архітектура гри: основні компоненти, сцени та менеджери

2.3. Реалізація шахових фігур: алгоритми руху

2.4. Реалізація механіки фізики: падіння, гравітація, взаємодія з об'єктами

2.5 Взаємодія з ігровим світом: гравець, ворог, обмеження і цілі

Розділ 3. Реалізація та тестування відеогри

3.1. Створення рівнів, шаблонів і варіативності

3.2. Розробка меню, налаштувань, переходи між рівнями

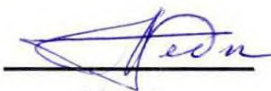
3.3. Аналіз результатів тестування та ефективності

7. Календарний план виконання роботи

№ п/п	Назва етапів кваліфікаційної роботи	Строк виконання етапів роботи	
		За планом	Фактично
1	2	3	4
1	Вибір теми, підбір літературних джерел, формування структури кваліфікаційної роботи	до 01.11.2024	01.11.2024 р.
2	I розділ (розкрити сутність і стан дослідження наукової проблематики роботи, після чого навести її загальну характеристику; огляд літератури за темою і вибір конкретних напрямів досліджень)	30.01.2025	30.01.2025 р.
3	II розділ (аналіз експериментальної частини про методики досліджень)	28.02.2025	28.02.2025 р.
4	III розділ (проведені теоретичні і (або) експериментальні дослідження та проектні рішення)	30.04.2025	30.04.2025 р.
5	Висновки	05.05.2025	05.05.2025 р.
6	Доопрацювання роботи з урахуванням зауважень наукового керівника	25.05.2025	25.05.2025 р.
7	Отримання відгуку керівника	25.05.2025	25.05.2025 р.
8	Отримання зовнішньої рецензії	28.05.2025	28.05.2025 р.

8. Дата видачі завдання «15» листопада 2024 р.

9. Керівник кваліфікаційної роботи


(підпис) Редич О.В.

10. Керівник освітньо-професійної програми


(підпис) Омельчук А.

11. Завдання до виконання отримав,
здобувач вищої освіти


(підпис) Пінчук К.С.

АНОТАЦІЯ

Розробка відеогри для розвитку логіки мовою C# на платформі Unity:

Мета кваліфікаційної роботи полягає у створенні відеогри, що поєднує логічні шахові механіки з фізичними взаємодіями у просторі, з використанням мови програмування C# та ігрового рушія Unity.

Предмет дослідження — використання шахових алгоритмів руху та фізичних властивостей для побудови логічних головоломок у середовищі Unity.

Об'єкт дослідження — процес створення двовимірної логічної гри.

За результатом дослідження створено і протестовано ігрову програму з навчальним потенціалом, що базується на класичних шахових правилах, поєднаних з фізикою падіння, платформами, взаємодією з перешкодами.

Гравець вирішує головоломки шляхом переміщення фігур відповідно до встановлених алгоритмів, досягаючи заданої мети.

Проведено повне тестування геймплею, логіки та механіки, що підтвердило стабільність гри та її ефективність як логічного тренажера

Результати кваліфікаційної роботи можуть бути використані як основа для подальших освітніх проєктів або комерційних логічних ігор.

Кваліфікаційна робота містить 61 сторінки, 19 рисунків, список використаних джерел з 49 найменувань, 2 додатки.

Ключові слова: логічна відеогра, Unity, мова програмування C#.

Рік виконання кваліфікаційної роботи: 2024–2025

Рік захисту: 2025

ABSTRACT

Development of a video game for the development of logic in C# on the Unity platform:

The goal of the qualification work is to create a video game that combines logical chess mechanics with physical interactions in space, using the C# programming language and the Unity game engine.

The subject of the research is the use of chess movement algorithms and physical properties to build logical puzzles in the Unity environment.

The object of the research is the process of creating a two-dimensional logical game.

As a result of the research, a game program with educational potential was created and tested, based on classical chess rules combined with the physics of falling, platforms, and interaction with obstacles. The player solves puzzles by moving pieces according to established algorithms, achieving a given goal.

A full testing of gameplay, logic and mechanics was carried out, which confirmed the stability of the game and its effectiveness as a logic simulator

The results of the qualification work can be used as a basis for further educational projects or commercial logic games.

The qualification work contains 61 pages, 19 figures, a list of used sources with 49 names, 2 appendices.

Keywords: logic video game, Unity, C# programming language.

Year of completion of the qualification work: 2024–2025

Year of defense: 2025

ЗМІСТ

ВСТУП.....	3
РОЗДІЛ 1. ТЕОРЕТИКО-МЕТОДОЛОГІЧНІ ОСНОВИ РОЗРОБКИ ЛОГІЧНИХ ВІДЕОІГОР	5
1.1. Відеоігри як інструмент розвитку логічного мислення	5
1.2. Особливості побудови логічних ігор: жанри, принципи, підходи.....	7
1.3. Шахові механіки у відеоіграх. Фізичні моделі в іграх.....	8
1.4. Технології розробки логічних ігор: огляд та обґрунтування вибору C# і Unity.....	11
1.5. Аналіз подібних ігор та рішень	13
РОЗДІЛ 2. СИСТЕМНО-АНАЛІТИЧНЕ ОБҐРУНТУВАННЯ АРХІТЕКТУРИ ГРИ.....	18
2.1. Моделювання ігрового процесу: логіка, цілі, рівні складності.....	18
2.2. Архітектура гри: основні компоненти, сцени та менеджери	19
2.3. Реалізація шахових фігур: алгоритми руху (кінь, тура, ферзь, пішак тощо)	21
2.4. Реалізація механіки фізики: падіння, гравітація, взаємодія з об'єктами	24
2.5. Взаємодія з ігровим світом: гравець, ворог, обмеження і цілі	26
РОЗДІЛ 3. РЕАЛІЗАЦІЯ ТА ТЕСТУВАННЯ ВІДЕОГРИ	30
3.1. Створення рівнів, шаблонів і варіативності	30
3.2. Розробка меню, налаштувань, переходи між рівнями	32
3.3. Оцінка ефективності гри як логічного тренажера	38
ВИСНОВКИ	42
СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ	44
ДОДАТКИ.....	48
Додаток А.....	48
Додаток Б	53

ВСТУП

Актуальність теми.

Зараз інформаційне суспільство стрімко розвивається в напрямку відеоігор. Ігри – це чудовий спосіб тренувати наш розум, і покоління міленіалів першим це усвідомлює. Всупереч застарілим переконанням, час, витрачений на Minecraft (чи будь-яку іншу гру), не витрачений даремно[39]. Загадки, голомолочки та ігри по типу «вгадай слово» з п'яти спроб, займають важливе місце у цій галузі. Якщо спробувати описати це одним словом, то це – ігри на логіку. Їх можна розглядати як джерело розваг та засіб для розвитку мислення. Такі ігри дозволяють навчитися моделювати складні проблеми, сприяють розвитку критичного мислення, вчать шукати нестандартні рішення та думати про наслідки своїх дій. Отже, відеоігри для розвитку логічного мислення – питання актуальне.

Об'єкт дослідження – відеогрові середовища як засіб розвитку аналітичного мислення.

Предмет дослідження – інструментарій розробки двовимірної логічної відеогри на мові програмування C# в ігровому рушії Unity.

Мета роботи полягає у створенні та аналізі відеогри, що сприяє розвитку логічного мислення користувачів, з використанням сучасного інструментарію – мови програмування C# і платформи Unity.

Для досягнення цієї мети було поставлено такі завдання:

- проаналізувати сучасний стан ігор, що спрямовані на розвиток логіки;
- вивчити особливості платформи Unity як інструменту для створення логічних ігор;
- дослідити можливості мови C# у розробці ігрових механік;
- реалізувати функціонал гри та протестувати його на практиці;
- сформулювати рекомендації щодо подальшого розвитку гри та її використання в освітньому процесі.

Методи дослідження.

У роботі використовуються: аналіз літератури та відео матеріалів, публікацій наукових досліджень по темі роботи, порівняльний аналіз існуючих аналогів, методи об'єктно-орієнтованого програмування, проєктування програмних систем, а також тестування функціоналу гри.

База дослідження.

Матеріали офіційної документації Unity, приклади реалізації подібних проєктів, а також наукові праці в галузі освітніх технологій та ігрової педагогіки.

Практичне значення роботи полягає в розробці власної гри, що поєднує механіки головоломок із можливостями Unity. Така гра створюється, з можливістю адаптації під різні рівні складності та типи навантаження.

Особистий внесок автора полягає у повному циклі розробки гри – від концепції до реалізації, включаючи аналіз джерел, вибір інструментів, проєктування логіки гри, її реалізацію, тестування та формулювання висновків і рекомендацій. Данна концепція гри є відображенням власного погляду автора на її потенційні можливості, щодо реалізації.

РОЗДІЛ 1. ТЕОРЕТИКО-МЕТОДОЛОГІЧНІ ОСНОВИ РОЗРОБКИ ЛОГІЧНИХ ВІДЕОІГОР

1.1. Відеоігри як інструмент розвитку логічного мислення

Відеоігри перестали бути тільки формою розваги. Тепер їх часто розглядаються як можливість для навчання дітей та власного розвитку, а технологічний прогрес дозволяє створювати ігри, в які не просто цікаво морнують час, а й виконують корисну дію. Окрім ігор на пам'ять та програм для вивчення іноземних мов, важливим напрямком є розвиток логічного мислення.

Під логічним мисленням мається на увазі критичний аналіз інформації, побудова логічних ланцюгів та встановлення причинно-наслідкових зв'язків. Це допомагає при участі в командних проєктах, програмуванні, математиці, плануванні, тощо.

Дослідження свідчать, що відеоігри певного типу дійсно можуть впливати на розвиток логіки [\[17\]](#). Найефективнішими у цьому аспекті вважаються ігри, в яких гравець змушений:

- приймати рішення на основі аналізу змінних умов;
- будувати гіпотези та перевіряти їх в дії;
- шукати нестандартні рішення;
- прогнозувати наслідки власних дій.

До таких ігор належать: головоломки, стратегічні симулятори, покрокові логічні ігри, шахи, задачі на просторове мислення та логічні задачі у формі квестів або сценаріїв де є безліч варіантів рішень, наприклад настільна гра в жанрі пригод «Підземелля та Дракони».

Для залучення важливо підтримувати інтерес до гри і інструментом для цього є **баланс** між ігровим процесом та інтелектуальним навантаженням. Якщо

гра надто проста, вона буде не цікавою. Якщо занадто складна або позбавлена чіткого зворотного зв'язку, інтерес до неї падає. Тому важливо створювати продукти, складність задач в яких збільшується поступово та адаптується до рівня гравця, а помилки стають приводом для прогресу[48].

Головоломки, стратегічні ігри, програмувальні задачі – це ігри та задачі що були винайдені для розвитку навичок, як просторове мислення, дедукція, пошук закономірностей та знаходження послідовностей. Такі виклики змушують гравця діяти в межах чітко визначених правил, але одночасно надають велику свободу дій в межах цих правил. Використання гейміфікації та можливостей комп'ютерних ігор є одним із сучасних підходів для покращення здібностей менеджерів, включаючи навички стратегічного мислення. Організації вже використовують відеоігри в сферах освіти, маркетингу, бізнесу та підприємництва, оскільки прогнозується, що навчання на основі ігор стане платформою для рефлексивного навчання з питань підприємництва[47].

Крім того, інтерактивність відеоігор дає змогу використовувати принцип "навчання через дію"[7], де гравець отримує досвід, експериментує, помиляється та вчиться на власних діях, що відрізняє відеоігри від пасивних форм навчання і робить їх надзвичайно ефективними та доступними. Гра дає значно краще засвоєння інформації, ніж книга чи фільм, бо в грі є вольова компонента. Тобі не просто розповідають про якусь ситуацію – ти сам змушений діяти в її умовах. Ти її не просто спостерігаєш, а проживаєш, є не пасивним, а активним учасником дії. Якщо якийсь сучасний вид мистецтва і вчить нас робити вибір та нести за нього відповідальність – то це саме вони[2].

У підсумку, відеоігри є ефективним і перспективним інструментом розвитку логічного мислення [5], особливо якщо їх структура, механіки та зміст будуються з урахуванням когнітивних цілей. Це відкриває великі можливості для поєднання

індустрії розваг та освіти, особливо в умовах стрімкої цифровізації навчального процесу.

1.2. Особливості побудови логічних ігор: жанри, принципи, підходи

Розробка логічних відеоігор вимагає як чіткого розуміння механізмів мислення, але не обов'язково потребує високого рівня технічної реалізації. До логічних ігор належать головоломки - пасьянси, кросворди, sudoku, а також багато антагоністичних настільних ігор[\[26\]](#), і саме це доводить їх просту реалізацію: досить листка паперу, з олівця, щоб зробити кросворд чи пограти в цятки. Незважаючи на зовнішню простоту, ігри цього жанру мають цілу низку характерних особливостей, які відрізняють їх від інших. У цьому підрозділі розглянуто основні типи логічних ігор, принципи побудови логічного геймплею та приклади таких ігор.

Під терміном «логічна гра» зазвичай мають на увазі, що основна активність гравця полягає у вирішенні задач на мислення, зокрема аналітичне, дедуктивне і в нашому випадку просторове.

Можна зазначити такі піджанри логічних ігор та їх приклади:

- *Класичні головоломки* — рішеннях завдань з однозначним або обмеженим числом рішень (наприклад: Судоку)
- *Логічні пригодницькі ігри* — мають сюжет та просуваються по ньому шляхом вирішенням задач (*Portal*);
- *Програмувальні ігри* — гравець повинен будувати послідовності дій та умов (*Human Resource Machine* та *Baba Is You*);
- *Фізичні головоломки* — де задачі побудовані на симуляції фізичних законів: гравітації, інерції, зіткнень[\[33\]](#) (*World of Goo*).

Частіше за все використовують змішання жанрів, де гравець повинен як мислити стратегічно, так і взаємодіяти з фізичним механіками середовища. Таких

ігор стає все більше, бо з розвитком технологій, які дозволяють легко реалізовувати фізику та інші механіки, порог входу в програмування стає все менше і менше.

Особливість логічних ігор полягає в тому, що акцент ставлять на необхідності виявляти закономірності, прогнозувати наслідки дій та приймати обдумані рішення, іноді не зважаючи на сюжет чи швидкості реакції. Такий тип взаємодії вимагає формальної структури, де кожна дія контролюється чітко визначеними правилами.

Логічним іграм необхідна система правил з внутрішніми обмеженнями, але свободою для експериментів. Ефективна логічна гра має пропонувати цікаве ігрове середовище[\[49\]](#).

Для створення логічних ігор хорошим початком буде врахування навчальної кривої зростання складності. Основою логічної гри є введення додаткових елементів, нових обмежень, в результаті гравець не лише проходить гру, а фактично навчається мислити в рамках все більш складніших логічних структур.

1.3. Шахові механіки у відеоіграх. Фізичні моделі в іграх

Шахи — одна з перших інтелектуальних забав, що з часом стали символом гри розуму. При чітким правилам, але великим можливостям для розвитку дій, шахові механізми стали привабливими для адаптації в цифровому світі, де їх застосовують як інструмент розвитку аналітичного мислення, планування та передбачення [\[12\]](#).

Тому шахові принципи часто використовуються як абстрактний шаблон, де будуються нові види логічної взаємодії.

Наприклад, у грі Into the Breach на (див. рис.1.1), шахові принципи втілені у вигляді покрокової тактики, де кожна одиниця має механіку можливості пересування та атаки, а гравець змушений прораховувати дії противника[\[32\]](#).

Окрім фантастичного дизайну, структура гри побудована на тих самих правилах: обмежений простір, послідовні ходи, чіткі правила. Into the Breach демонструє, як шахи можуть бути трансформовані у новий контекст, зберігаючи ключові принципи.



Рис. 1.1 Into the Breach

Хоча в індустрії багато схожих ігор, експериментальні проєкти демонструють перспективність цього напрямку. Для перших експериментів з природньою мовою, були обрані саме шахи[44]. Ідея шахової головоломки в середовищі з гравітацією є доволі цікавою і ще не набула широкого поширення, тому цю тему буде розглянуто в межах даної кваліфікаційної роботи.

Шахові механіки залишаються гнучким інструментом для створення інтелектуальних задач у відеоіграх. Глибина та обмеження простору, створюють умови для побудови рівнів. Розширення класичних механік через додавання фізики відкриває нові можливості для геймдизайну та користатися зрозумілим інтерфейсом для швидкого старту. Данна робота — це гра, де класичні шахові принципи поєднуються з елементами фізичної симуляції, створюючи унікальну взаємодію.

Фізика у відеоіграх — це засіб реалістичного зображення руху об'єктів та механіка для взаємодії гравця із середовищем. У старих або настільних логічних іграх, де вся поведінка об'єктів працює за правилами 2D простору, фізично орієнтовані ігри моделюють додатковий шар реальності. З розвитком обчислювальних потужностей та появою фізичних рушіїв типу *NVIDIA PhysX*, *Box2D* або *Unity Physics* стали можливими складні взаємодії між об'єктами у просторі з гравітацією.

Фізична модель в іграх — це набір алгоритмів та опис руху та взаємодію тіл у 3D. У більшості ігор, що побудовані на рушії Unity, використовується механіка Ньютона. З точки зору комп'ютерної графіки, це означає розрахунок векторів сили, маси, швидкості, прискорення у реальному часі.

Основу такої моделі становлять наступні поняття:

- Тіла з масою — об'єкти, що мають інерцію та піддаються впливу сил.
- Сили — гравітація, імпульси, опір середовища, які впливають на поведінку тіл.
- Колізії — обробка зіткнень між об'єктами за допомогою колайдерів, що визначають геометрію взаємодії.
- Рух — залежність швидкості та прискорення від прикладених сил, що забезпечує реалізм динаміки. [\[24\]](#)

Рушій Unity має вбудовані фізичні модулі, що дозволяють зробити ці ефекти без глибокого програмування.

Unity використовує інтеграцію з рушієм *NVIDIA PhysX*, що надає можливості для симуляції твердих тіл, виявлення зіткнень та інерції.

Хоча фізичні моделі зазвичай асоціюються з динамічними жанрами, останні десятиліття продемонстрували успішну інтеграцію фізики у логічні ігри. В іграх типу *World of Goo*, *Portal*, *Cut the Rope* або *Crayon Physics Deluxe*. Фізичні властивості стали основою механіки мислення: рішення задачі передбачає

розуміння траєкторій, впливу гравітації, моментів сили або балансу об'єктів. Саме через добре подану ідею такої головоломки *Portal* в свій час отримав всесвітню популярність[\[43\]](#).

Для побудови складних, інтерактивних завдань, фізичні моделі відкривають нові горизонти у шахових механікаї, що робить класичну ігрову структуру, основоб для чогось більшого.

1.4. Технології розробки логічних ігор: огляд та обґрунтування вибору C# і Unity

Ринок інструментів для \розробки ігор має як прості конструктори так і повноцінні рушії, якими користуються корпорації для своїх розробок. Для створення гри важлива потужність рушія та висока керованість логікою, структурою гри, а також підтримка зручної мови програмування.

Unity на сьогодні є однією з найпопулярніших платформ у світі ігрової розробки. За даними авторитетних джерел та офіційної статистики, близько 60-70% найприбутковіших мобільних ігор у світі створені саме за допомогою Unity [\[28, 34\]](#), а сама платформа обслуговує близько 3 мільярдів пристроїв по всьому світу [\[46\]](#).

Unity підтримує як 2D, так і 3D-розробку [\[11\]](#), забезпечуючи розробника спектром інструментів: від візуального редагування сцени до роботи з фізичними компонентами та анімацією. У логічної гри з фізикою, особливо цінним є *2D Physics Engine* на бібліотеці *Box2D*, завдяки якому можливе падіння, зіткнення та блокування.

Перевагою Unity є його компонентна архітектура. Кожен об'єкт у грі — є окремим, що може приєднувати набір компонентів: від колайдера і фізичного тіла до скриптів логіки. Це спрощує налаштування логіки фігур та законів фізики.

Для розробника-початківця Unity має базу шаблонів, форумів і курсів, що дає середовище, де більшість проблем вже були вирішені спільнотою, а для

досвідченого розробника це платформа для мистецтва. Завдяки цьому Unity активно використовується в комерційній розробці зазвичай в інді-іграх та навчальних цілях по типу проектів з освітніми іграми або науковими симуляціями [28].

За даними авторитетних джерел, навіть попри падіння C# в рейтингах він залишається в топ 5 [20] та є сновною мовою програмування для Unity (див. рис. 1.2). Це високорівнева об'єктно-орієнтована мова, розроблена компанією Microsoft. Її синтаксис є досить строгим, що зменшує кількість помилок у коді, а багата стандартна бібліотека дозволяє ефективно працювати з логікою, колекціями, обробкою подій, даними, UI та мережею [8]. Саме C# забезпечує модульність та підтримку проєкту, а також спадкування та поліморфізм — механізмів для створення добре структурованої гри.

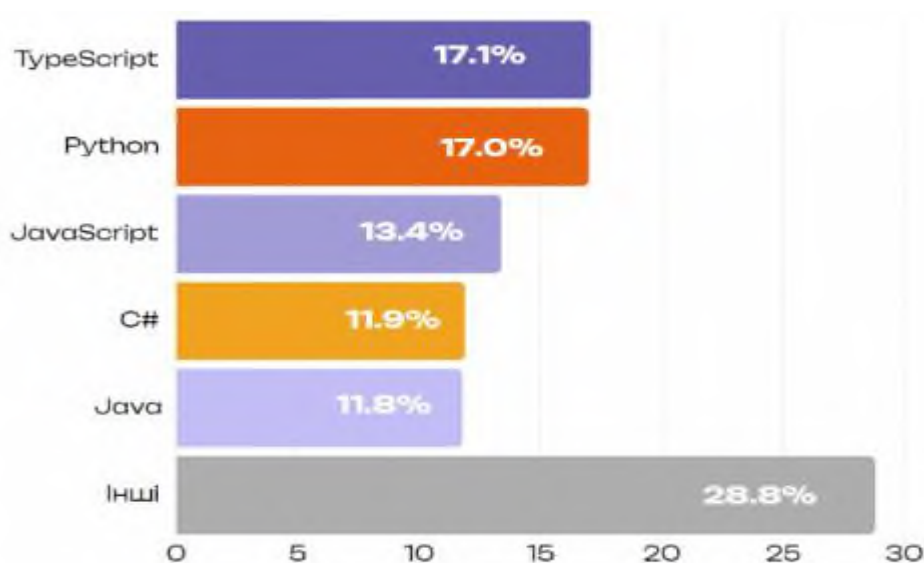


Рис. 1.2. Рівень популярності мов програмування на 2025 рік [19]

Вибір Unity як рушія для гри був логічно обґрунтованим: це платформа, яка дозволяє інтегрувати логіку, фізику, графіку та UI в єдину систему, а мова C# забезпечує достатню гнучкість для побудови архітектури гри. У сукупності це створює умови для розробки шахової шоловоломки з елементами гравітації.

1.5. Аналіз подібних ігор та рішень

Розробка логічної гри, на шахових правилах із додаванням фізичних взаємодій, передбачає ознайомлення з існуючими прикладами. Такий аналіз виявляє ефективні дизайнерські рішення та проблемні зони, що виникають у процесі розробки ігор зі схожим геймплеєм.

Першим прикладом цифрової реалізації шахової логіки є класичні шахові симулятори **Chessmaster** або **Lichess** (див. рис. 1.3-4), які дають можливість грати проти комп'ютера або інших гравців онлайн [31]. Комп'ютери мають високий рівень точності, зосередженістю на обчислювальних стратегічних алгоритмах і відсутністю фізичних обмежень, тому користуються популярністю серед гройсмейстерів.

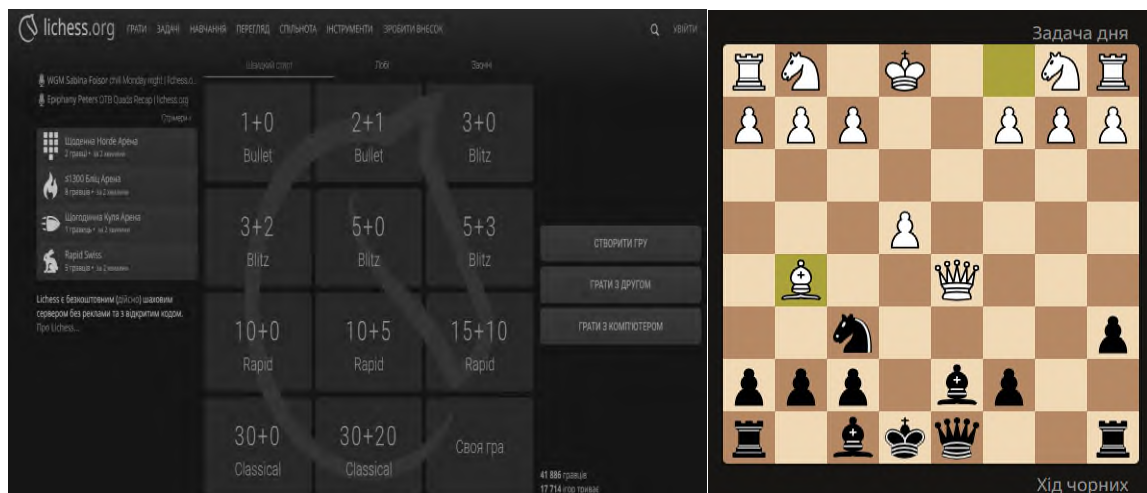


Рис. 1.3-4 Геймдизайн Lichess

Перевагою симуляторів є простота використання, однак у контексті креативного мислення вони мають потенціал для розвитку. Ігровий простір жорстко обмежений, а кожна фігура має лише класичне функціональне призначення, які обмежують гнучкість, хоча мають свою привабливість.

Ігри відходять від традиційної форми, зберігаючи ідеї. Фігури з унікальними правилами руху, обмежене ігрове поле та необхідність передбачення ходів. Одним із прикладів такої трансформації є "**Pawnbarian**" (2021) — покрокова рогалик-

головоломка, що використовує шахові фігури як картки дій на полі бою [3]. Гра переносить знайомі патерни у контексті бою, створюючи новий тип взаємодії на базі класичної логіки та є прикладом поєднання карткової гри з шахами — двох найпопулярніших видів настільних розваг.

"5D Chess With Multiverse Time Travel" (див. рис.1.5) це класичні шахи з механіками паралельних вимірів і переміщення у часі. Це дає змогу змінювати історію ходів і грати на кількох дошках одночасно [4]. Хоча ця гра не має фізичного рушія, вона базується на концепції шахових ігор з додаванням альтернативних систем мислення, у даному випадку часової логіки.

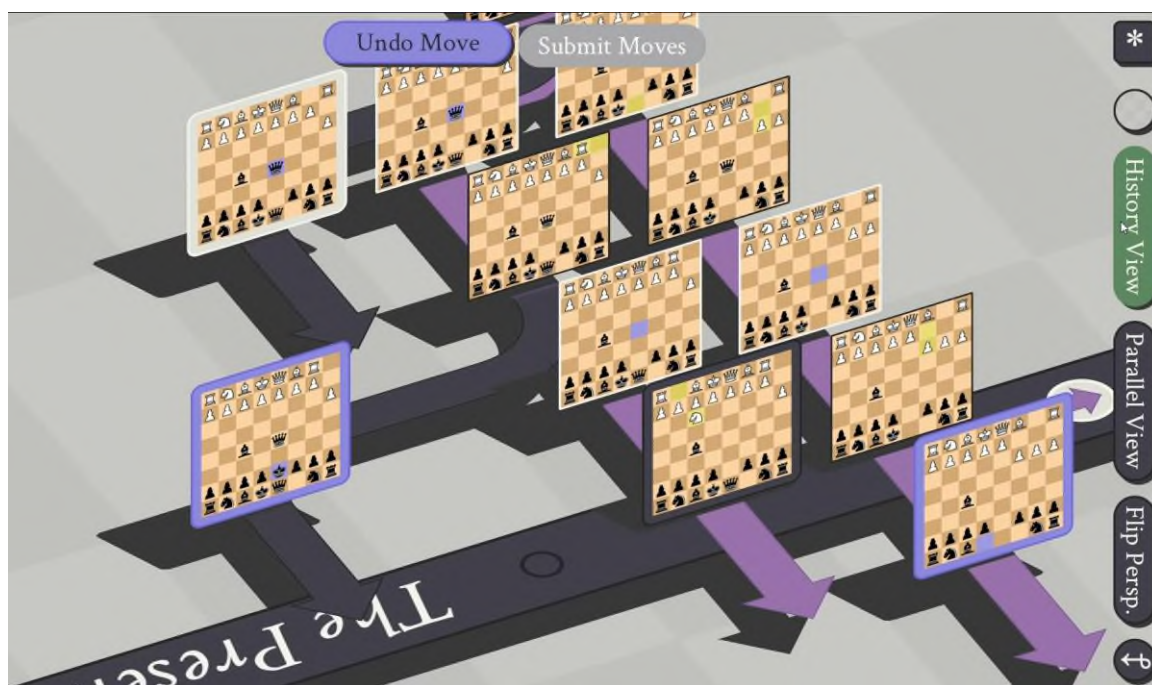


Рис.1.5 Ігрове середовище "5D Chess With Multiverse Time Travel"

Відомими прикладами ігор на фізичних взаємодіях з логічним спрямуванням є "World of Goo", "Human: Fall Flat" (див. рис.1.6) "Portal", де фізичні взаємодії (гравітація, імпульс) є не декорацією, а центральною ігровою механікою.



Рис. 1.6. Human: Fall Flat

У контексті розроблюваної гри фізика інструмент, що виконує роль модифікатора, змінюючи спосіб інтерпретації правил. На прикладі цих проєктів можна побачити, що фізика може виступати як основна складова логічної задачі, що може змінити погляд і на звичайні шахи.

Висновки з аналізу

Огляд існуючих реалізацій вказує на те, що на перетині шахової логіки та фізичних механік на сьогодні існує велика кількість ігор, хоча більшість залишаються в межах чистої шахової симуляції або повністю відходять від формальної логіки на користь іншим ігровим механікам.

Висновки до Розділу 1

У першому розділі було виконано теоретико-методологічний аналіз фундаментальних основ, які лежать в основі створення логічних ігор, з'ясовано роль відеоігор як інструменту розвитку логічного мислення, а також обґрунтовано вибір інструментарію,

У підрозділі 1.1 встановлено, що логічні ігри мають значний потенціал для розвитку мислення, прогнозування та навичок вирішення задач. Завдяки інтерактивності та наявності структури правил, вони створюють унікальне середовище, в якому гравець здобуває знання через дію. Відеоігри дедалі частіше використовуються у навчальному процесі як засіб мотивації та активний компонент розвитку мислення.

У підрозділах 1.2 визначено жанрові особливості логічних ігор та охарактеризовано шахові механіки як ефективну модель формалізованої логіки, яка має потенціал до адаптації в нових формах. Проаналізовано, яким чином шахова логіка може бути використана у побудові просторових задач.

У підрозділі 1.3 висвітлено теоретичні аспекти застосування фізичних моделей у відеоіграх. Було показано, що симуляція гравітації, зіткнень і просторового розташування дозволяє збагачувати логічні задачі додатковими рівнями складності, формуючи динамічні ситуації. Фізика в ігровому контексті виступає як фактор, що активізує формальну логіку, а отже — є важливою складовою освітнього потенціалу гри.

У підрозділі 1.4 обґрунтовано вибір платформи розробки. Рушій Unity був обраний завдяки його гнучкості, розвиненій 2D-фізиці та наявності навчальних прикладів від широкої спільноти.

У підрозділі 1.5 проведено аналіз сучасних ігор, що поєднують шахові правила, фізичну симуляцію та логіку. Було виявлено, що реалізації подібного підходу вже існують та досі мають свого гравця. Це підтверджує актуальність обраного напрямку дослідження.

Висновок з теоретичного аналізу доводить, що поєднання шахових механік із фізичною взаємодією в межах одної ігрової моделі має пізнавальну та інженерну цінність. Застосування інструментів Unity і C# дає фундамент для реалізації гри в рамках кваліфікаційної роботи. У наступному розділі буде

здійснено системно-аналітичне обґрунтування структури гри, її логіки та алгоритмів, що забезпечують досягнення освітньої мети.

РОЗДІЛ 2. СИСТЕМНО-АНАЛІТИЧНЕ ОБҐРУНТУВАННЯ АРХІТЕКТУРИ ГРИ

2.1. Моделювання ігрового процесу: логіка, цілі, рівні складності

Процес моделювання ігрового середовища є основною задачею при розробці логічної відеогри, оскільки саме він визначає характер взаємодії користувача з системою, структуру завдань, послідовність викликів, які постають перед гравцем, а також умови досягнення ігрових цілей[13]. Конкретно в цьому випадку, для інтелектуального проєкту, що поєднує елементи шахової логіки з фізичним моделюванням, постає необхідність у побудові детальної поведінки ігрових об'єктів, визначенні ігрової мети та формалізації рівнів складності.

Основна логіка гри лежить в принципах шахової механіки. Гравцеві надається контроль над фігурами одного кольору, які пересуваються відповідно до класичних шахових правил: кінь рухається буквою «Г», тура — по прямій, пішак — уперед тощо. Водночас ігровий простір доповнюється елементами фізичної взаємодії: відсутність опори під фігурою призводить до її падіння, фігури можуть стояти на рухомих або статичних платформах, взаємодіяти з перешкодами та іншими об'єктами середовища.

Ця гібридна модель передбачає, що кожен хід фігури несе потенційні ризики — неправильне розміщення може призвести до втрати позиції через падіння, що поже зартимати проходження рівня або навіть зробити його неможливим. Таким чином, гравець має аналізувати не лише шахову позицію, а й геометрію рівня та шукати послідовність дій що приведе до перемоги за найменшу кількість кроків. Внаслідок цього досягнення головної цілі — доступу до ворожого короля вимагає використання логічного мислення та просторового уявлення.

Основні цілі та рівні складності

Метою гри є досягнення ворожого короля, розташованого на віддаленій частині рівня. Залежно від рівня, шлях до цієї мети може включати проміжні завдання:

- уникнення фізичних пасток;
- застосування кількох фігур для відкриття шляху;
- використання об'єктів середовища для створення опори;
- знищення перешкод;
- збереження однієї ключової фігури до кінця рівня.

Ігрові події запускаються у кодї гри при досягненні певного стану: контакт із ворожим королем, активація плитки, знищення фігури, падіння об'єкта тощо. Для підтримання інтересу гри, реалізовано систему рівнів із поступовим ускладненням. Рівні класифікуються за параметрами:

1. Кількість та тип фігур: початкові рівні обмежуються однією фігурою.
2. Структура рівня: від простих, лінійних карт до складних, розгалужених, з рухомими платформами.
3. Фізичні виклики: на пізніх рівнях з'являються платформи, об'єкти, що змінюють стан (наприклад, активуються під вагою фігури).
4. Неочевидність рішень: деякі рівні вимагають виконання дій по порядку, часто передбачаючи жертву фігурою для досягнення перемоги.
5. Обмежені ресурси: ускладненням обмеженням кількості ходів.

Таким чином, гра виконує розважальну функцію та допомагає з розвитком інтелектуальних здібностей, що відповідає її темі як логічної освітньої гри.

2.2. Архітектура гри: основні компоненти, сцени та менеджери

Архітектура гри — це програмна структура та побудова взаємозв'язків між внутрішніми компонентами, яка забезпечує роботу і цілісність ігрового процесу. У випадку проекту, що поєднує класичну шахову логіку з фізичною взаємодією

об'єктів, встає потреба у створенні гнучкої, модульної архітектури, яка дозволяє виконувати обробку дій фігур та масштабувати гру за рахунок нових сцен, механік і рівнів без необхідності глобальної перебудови коду.

У цій грі архітектура побудована за принципами розділення відповідальності (*Separation of Concerns*) [16] та слабкої зв'язності компонентів, що є ключовими для підтримуваних програмних систем. В основі лежить структура на базі ігрового рушія Unity, у якій застосовується патерн "менеджерів" — спеціалізованих класів, які координують певні аспекти гри (управління станами, завантаженням сцен чи обробкою логіки).

Кожен рівень гри реалізовано у вигляді окремої сцени. Яка відповідає за конкретну головоломку з унікальним розташуванням фігур і перешкод.

Для управління переходом між сценами використовується централізований **BoardManager**, який забезпечує завантаження, оновлення інтерфейсів та обробку подій при перемиканні.

Принципом побудови ігрових об'єктів в Unity — це композиційна модель [26]. Кожен об'єкт складається з набору компонентів, що виконують функції: пересування, взаємодію з фізикою, обробку коду.

Фігури реалізовані як окремі класи-нащадки базового класу *Piece*, який задає основні властивості: тип, колір, правила руху. Специфікація руху відбувається через перевизначення методів, що визначають допустимі клітинки для ходу з урахуванням фізичних обмежень.

Ключову роль в побудові гри відіграє **GameManager** (Додаток Б, ст. 56) — центральний координаційний компонент, відповідальний за запуск гри, реєстрацію фігур, перевірку переможних умов, а також відстеження активної фігури [36]. *GameManager* взаємодіє з іншими системами — *UIManager* для оновлення інтерфейсу, *LevelManager* для старту рівня та *InputManager* для моніторингу дій гравця.

Застосування менеджерів дозволяє уникнути великої кількості неструктурованого коду в різних скриптах і створює чітку ієрархію залежностей. Важливо, щоб кожен менеджер відповідав лише за одну сферу логіки, що повністю узгоджується з принципами *SOLID* та *Model-View-Controller* у геймдизайні [9].

Інтеграція фізики у шахову механіку відбувається через компоненти *Rigidbody* та *Collider*, які контролюють гравітацію, зіткнення та падіння об'єктів. Для динамічного аналізу стабільності платформи застосовується метод *MovePiece*, що дозволяє визначити наявність опори під фігурою. Якщо опора відсутня — запускається анімація падіння, і фігура вважається втраченою.

Це накладає додаткові вимоги на архітектуру: вона повинна дозволяти об'єктам реагувати на зміни середовища в реальному часі, що досягається через послідовно-організовану систему повідомлень, що дозволяє фігурам, платформам та менеджерам обмінюватися сигналами без прямого зв'язку.

Такий вид архітектури дозволяє легко додавати нові фігури, типи рівнів, механіки взаємодії без модифікації існуючих модулів. Вона не лише відповідає вимогам до ігрової функціональності, а й забезпечує масштабованість, що спрощує розуміння коду та дає можливості для потенційного розширення гри — наприклад, у напрямку багатокористувацької гри або створення редактора рівнів.

2.3. Реалізація шахових фігур: алгоритми руху (кінь, тура, ферзь, пішак тощо)

Оскільки центральною механікою гри є переміщення фігур за класичними шаховими правилами, реалізація коректної логіки руху кожної з них постає як ключова функціональна задача. В межах запропонованої відеогри фігури використовуються не для шахової партії, а як інструмент для вирішення головоломок із просторовими та фізичними обмеженнями. Проте точно

дотримання їхніх канонічних можливостей руху слугує фундаментом ігрової логіки та ключом до швидкої адаптації гравців.

Кожна фігура в архітектурі реалізована як окремий об'єкт, що наслідує загальний абстрактний клас *BoardManager*. Цей клас визначає спільні властивості, наприклад координати на сітці, колір, статус активності та посилання на фізичні компоненти (*Rigidbody*, *Collider*). Основним методом, реалізації логіки є *OnMouseDown*, що повертає список допустимих клітинок для переміщення згідно з правилами.

Приклади реалізації алгоритмів руху фігур

Кінь

Алгоритм руху коня має один з найбільш складних алгоритмів руху, оскільки жодна інша фігура не здійснює стрибків. Рух коня реалізується як набір із 8 можливих точок координат: $(\pm 2, \pm 1)$, $(\pm 1, \pm 2)$. На відміну від інших фігур, кінь не враховує перешкод, оскільки за правилами його хід рахується як стрибок. Перевірка допустимості поля включає перевірку меж поля та відсутності фігур власного кольору. (Додаток А, ст. 51)



Тура (ладья)

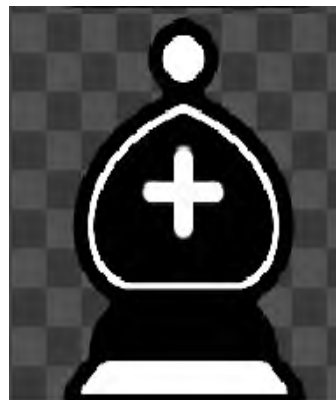
Тура пересувається до першої перешкоди. Алгоритм базується на просуванні по кожному з чотирьох напрямків: вгору, вниз, вліво, вправо до меж поля або до зіткнення з фігурою. Якщо



фігура супротивника, рух дозволено включно з цією клітинкою. При виявленні будь-якої фігури шлях у цьому напрямку завершується. (Додаток А, ст. 49)

Слон

Рух по діагоналях реалізується аналогічно до тури, але напрямки обмежено діагоналями: $(\pm 1, \pm 1)$. Починає гру на полі певного кольору, але навідміну від звичайних шахів може змінювати колір клітинки за допомогою мехоніки падіння. Для виявлення обмежень алгоритм виконує поступове просування по діагоналі з перевіркою на кожному кроці. (Додаток А, ст. 48)



Ферзь

Ферзь поєднує властивості тури та слона, маючи змогу пересуватися у всіх восьми напрямках. Алгоритм має обмеження — цикл по всіх напрямках з просуванням до межі або перешкоди. Така модель дозволяє легко масштабувати логіку для додаткових фігур. (Додаток А, ст. 50)



Пішак

Пішак має найбільш складну поведінку. У стандартному шаховому контексті пішак рухається вперед на одну клітинку, а з початкової позиції — на дві. Він може атакувати по діагоналі, якщо цільова клітинка зайнята ворожим об'єктом [\[15\]](#). (Додаток А, ст. 51)



2.4. Реалізація механіки фізики: падіння, гравітація, взаємодія з об'єктами

Однією з унікальних характеристик досліджуваної гри є поєднання логіки шахових правил із законами фізики, що ускладнює сприйняття шахових фігур і розширює ігрові можливості. Саме включення гравітації, падіння з платформ і взаємодії з навколишніми об'єктами формує механіку вирішення головоломок.

Для досягнення реалістичного падіння у грі використовується вбудований фізичний рушій Unity – *PhysX*, який дозволяє моделювати силу тяжіння. Всі рухомі об'єкти, мають компоненти *Rigidbody2D* — та налаштовані на взаємодію з *BoxCollider2D* [\[35\]](#).

Приклад налаштування :

```
void Awake()
{
    // Налаштовуємо Rigidbody2D
    rb = GetComponent<Rigidbody2D>();
    if (rb == null)
        rb = gameObject.AddComponent<Rigidbody2D>();

    rb.bodyType = RigidbodyType2D.Static;
    rb.gravityScale = 1f;
    rb.constraints = RigidbodyConstraints2D.FreezeRotation;

    // Додаємо колайдер
    var col = GetComponent<BoxCollider2D>();
    if (col == null)
```

```

gameObject.AddComponent<BoxCollider2D>();
}

```

Ключова умова для фізичної симуляції — це зняття обмежень на рух. До ходу фігури об'єкти залишаються в статичному режимі, не навантажуючи фізичний можуль. Після виконання ходу об'єкт переходить у динамічний стан, що дозволяє відчувати результат взаємодії із середовищем, наприклад через падіння за відсутньої платформи або зміщення іншої фігури).

Для всіх фігур враховується стан поля та наявність платформи під фігурою. Якщо опори немає або вона перебуває у нестабільному стані, скрипт активує гравітацію, що означає початок падіння. Якщо до кінця ігрової зони по осі Y немає перешкоди, фігура знищується. Так синхронізована шахова логіка з фізичними обмеженнями ігрового середовища.

На етапі розрахунку допустимих ходів використовується перевірка через *OnCollisionStay2D* та *OnTriggerEnter2D*. У такий спосіб досягається поєднання логіки шахів та актуального стану сцени, що є критично важливим у динамічному середовищі.

Гравітаційна сила в сцені налаштована відповідно до фізичних стандартів Unity (9.81 м/с^2 вниз по осі Y)[\[41\]](#), але за потреби адаптується для досягнення балансу. Наприклад, щоб зробити падіння зрозумілішим і менш миттєвим, було зменшено коефіцієнт гравітації в компоненті *Physics.gravity* за такою формулою:

Інтерактивні елементи ігрового поля, такі як нестабільні платформи, кнопки або перемикачі реалізовано через тригерні колайдери та анімації станів. Наприклад, при наступанні фігурою на певну плитку може запускатись анімація, що поступово руйнує деякий об'єкт чи рухає предмет, після чого він падає вниз. Об'єкти також реагують один на одного. При падінні тіла відбувається зіткнення згідно з заданими колайдерами.

Для забезпечення передбачуваності гри, застосовується контрольований підхід до активації фізики. Більшість об'єктів не можуть діяти один на одного,

окрім випадків задіяних через код по типу платформи, що активуються масою. Це реалізується шляхом активації взаємодії між колайдерами або активації фізичних шарів.

Крім того, використовується *MovePiece* для перевірки наявності опори під фігурою перед і після ходу. Це дозволяє точно визначити момент падіння і відповідно реагувати на зміну ігрового стану — наприклад, завершення рівня або поразку при втраті ключової фігури.

Фізична складова гри виступає технічним елементом та виконує логічну функцію. Завдяки фізичній моделі виникає багатовимірний просторовий динамізм, який змушує гравця мислити не лише над тактикою, а й над наслідками своїх дій, оскільки кожна фігура має свої правила та може застрягти у начебто відкритому просторі. Механіка падіння інколи тільки заважає гравцю, але може бути використана як інструмент для активації певних подій або очищення шляху до цілі.

2.5. Взаємодія з ігровим світом: гравець, ворог, обмеження і цілі

Гравець у запропонованій грі не використовує конкретну фігуру, а керує усіма фігурами одного кольору (аналогічно до класичних шахів), приймаючи рішення щодо пересування та порядку дій.

Фігура може бути обрана кліком й активує інтерфейс можливих ходів, який будується на основі алгоритмів (описаний у попередніх підрозділах).

Гравець має змогу діяти довільно, але на пізніх рівнях може вводитись обмеження на кількість ходів, часовий ліміт або пастки. Це підвищує навантаження, змушуючи оцінювати ситуацію на кілька кроків наперед.

Основною ігровою ціллю у кожному рівні є досягнення позиції, на якій розташовано **ворожого короля**. Проте і сам ворог має можливості для атаки за

правилами шахів. Його присутність маркує кінець рівня, і саме це символізує успішне завершення логічного завдання.

Однак логічно, що в деяких варіаціях рівнів можуть з'являтися додаткові перешкоди та ворожі об'єкти (див. рис.2.7) що заважатимуть досягненню цілі, наприклад, інші фігури, які блокують шлях, або рухомі платформи, які треба активувати. Такий підхід дає змогу уникнути традиційного конфлікту, натомість акцентуючи увагу на головоломці та просторовій навігації.

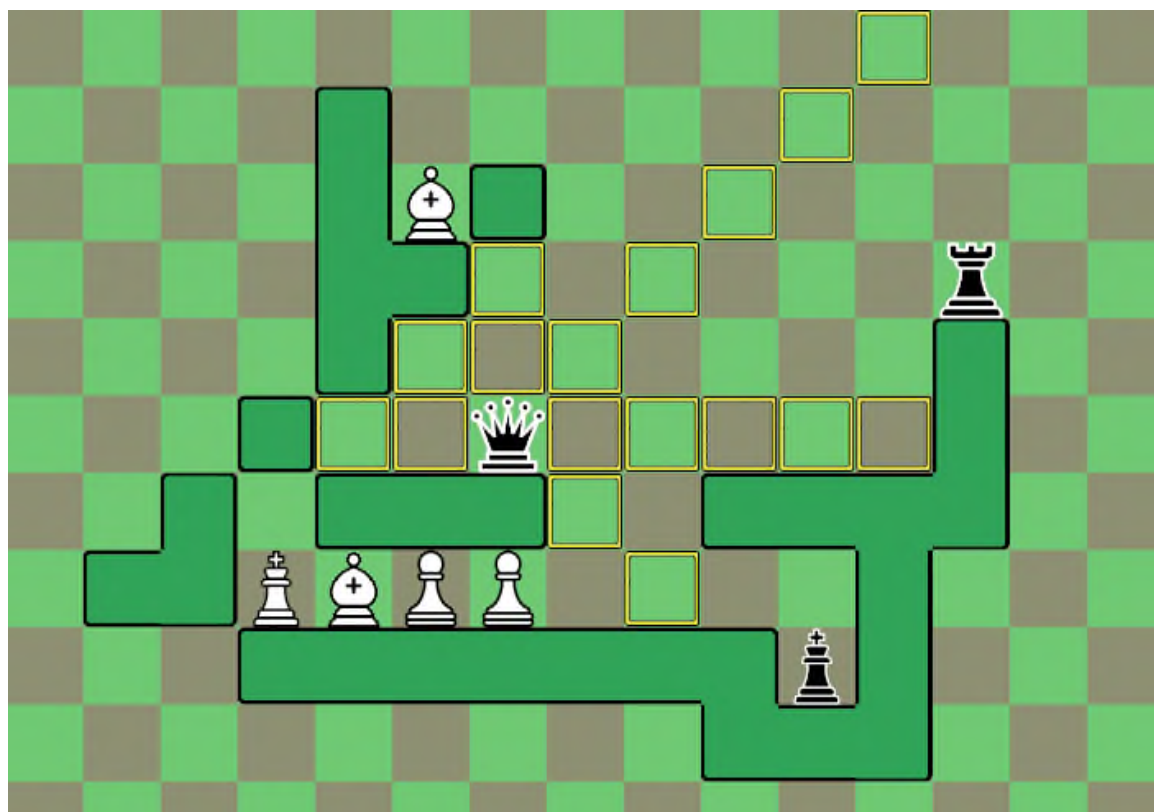


Рис.2.1. Приклад можливого ворожого об'єкту

Обмеження гравця є невід'ємною частиною ігрової логіки. Вони включають:

Обмеження ходів: кожна фігура має визначені закономірності, що виключає випадковість і потребує планування.

Фізичні межі рівня: простір обмежений платформами, перешкодами, пустотами, через які фігури можуть впасти.

Гравітаційні пастки: падіння в пустоту прирівнюється до втрати фігури.

Такі обмеження виконують функцію умови задачі та правил, що стимулює гравця до логічного аналізу та побудови сценаріїв рішень.

Метою гравця не завжди є фізичне знищення ворога, але завжди логічне досягнення позиції, при якому певна фігура потрапляє на клітинку розташування ворожого короля. Досягнення цілі є кульмінацією рівня і вважається проявом перемоги над задачею.

Висновки до розділу 2

У другому розділі кваліфікаційної роботи було здійснено всебічний аналіз і практичне моделювання основних етапів розробки логічної відеогри, що базується на принципах класичних шахів, поєднаних із механікою фізичної взаємодії. В основі підходу лежить ідея створення логічної гри у якій гравцю необхідно адаптуватися до умов ігрового світу, дотримуючись правил, знайомих із класичної гри, та розв'язувати нестандартні задачі, у середовищі з гравітацією.

Проведене моделювання ігрового процесу зазначило важливість опрацювання логіки фігур, цілей гравця та багаторівневої системи складності. Були визначені основні компоненти проєкту, такі як: сцени, контролери, менеджери логіки та фізики. Такий підхід реалізує модель рівнів, в якій кожен елемент — має визначене місце та призначення.

Алгоритми руху шахових фігур розроблялися з урахуванням як класичних шахових правил, так і обмежень фізичного середовища. Ці алгоритми дозволяють гарантувати, що кожна дія гравця залишається в межах допустимого логічного простору, тому сприяє формуванню коректного ігрового досвіду. Реалізація механіки гравітації та падіння створює складності, що перевіряють підготовку та передбачливість гравця.

Окремо розглянуто питання зворотного зв'язку. Система перевірки коректності ходів, досягнення мети й обробки окремих ситуацій є основою, на

якій будується довіра користувача до логіки гри. Ця система забезпечує стабільність та підсилює відчуття закінченості після проходження кожного рівня.

Узагальнюючи викладене, можна стверджувати, що реалізована концепція гри є логічно структурованою та обґрунтованою.

РОЗДІЛ 3. РЕАЛІЗАЦІЯ ТА ТЕСТУВАННЯ ВІДЕОГРИ

3.1. Створення рівнів, шаблонів і варіативності

Однією з фундаментальних складових структури логічної відеогри, орієнтованої на розвиток аналітичного мислення, виступає побудова рівнів, що поступово ускладнюються та водночас зберігають баланс між викликом та доступністю. У контексті запропонованої гри, яка базується на шахових принципах із додаванням механіки фізики, проєктування рівнів потребує не лише геометричної композиції, але й методичної побудови логічних сценаріїв, які спонукають гравця до дедукції, експериментування та поступового набуття стратегічного мислення.

Кожен ігровий рівень розглядається як навчальний модуль, що репрезентує певний клас логічних задач. Основним принципом побудови рівнів виступає поступовість: від простого керування однією фігурою без перешкод — до складних композицій з кількома об'єктами, які взаємодіють між собою в обмеженому просторі.

У процесі розробки системи рівнів було вирішено впровадити **механізм шаблонів**. Під шаблоном розуміється прив'язки до деяких параметрів, а абстрактна структура, що описує логіку [1] рівня: відокремлюючи рівень презентації від рівня даних

Завдяки шаблонам стає можливим:

- швидко створювати велику кількість рівнів, за готовими моделями;
- забезпечити цілісність основного;
- автоматизувати тестування нових конфігурацій рівнів, використовуючи базовий сценарій взаємодій.

У практичній реалізації шаблони зберігаються у вигляді JSON-файлів, що описують розміщення об'єктів, типи фігур, властивості плиток, зони гравітації та стартові умови[30]. Під час запуску рівня рушій Unity завантажує відповідний шаблон і динамічно будує сцену згідно з його параметрами.

Окрім цього, у перспективі планується реалізація **редактора рівнів**, що дозволить створювати власні рівні за допомогою графічного інтерфейсу та значно підвищить рівень залучення користувачів яким вже подобається такий вид ігор.

Побудова редактора рівнів

У процесі розробки логічної гри, важливою є розробка редактора рівнів, який дозволяє створювати ігрові сцени без необхідності втручання у програмний код. У контексті розробки проєкту на базі Unity, редактор рівнів буде зручним інструментом для розробника та основою для швидкого зросту та розширення функціональності у майбутньому. Такий інструмент буде корисний, оскільки скористатись ним можна навіть без доступу до кода звичайним гравцем. Що дозволить створювати карти власноруч, це також корисно для розвитку аналітичного мислення, адже побудувати цікаву карту набагато складніше ніж пройти її.

Редактор рівнів у грі має бути внутрішньоігровим інструментом, який дозволяє створювати нові сцени, розставляти фігури, перешкоди та інші об'єкти ігрового середовища безпосередньо через графічний інтерфейс. Основна ідея полягає у тому, що розробник або дизайнер може «намалювати» логіку рівня у візуальній формі, яка автоматично конвертується у внутрішнє представлення (структуру даних), придатне для збереження, обробки й повторного завантаження.

Розробка редактора базувалась на таких принципах:

- максимальна інтуїтивність;
- відокремлення логіки створення сцени від логіки проходження;

- візуальний зворотній зв'язок у режимі реального часу.

Редактор складається з трьох функціональних блоків: **інтерфейс побудови рівня, механізм збереження/завантаження, візуалізація ігрового світу** [6].

У першому блоці реалізовано інструменти вибору фігур, типів клітин (звичайна, платформа тощо), встановлення позиції гравця і ворога. Для забезпечення взаємодії використовуються UI-елементи Unity — кнопки, палітри, панелі інструментів.

Другий блок відповідає за трансляцію сцени. Завдяки використанні кореневої файлу з усіма скриптами проекту та UI дизайном включно. Можна окремо створювати сцени з рівнями та підвінтажувати функціонал.

Третій блок забезпечує миттєвий візуальний результат: будь-яка дія в редакторі миттєво відображається на сцені, що прискорює процес розробки та дозволяє виявляти помилки або конфлікти об'єктів.

Наприклад, при додаванні нових типів об'єктів, фігур або сценаріїв поведінки, достатньо зареєструвати їх у основній системі редактора для використання у майбутній сцені. Це дозволяє у перспективі реалізувати інструмент для творчості.

3.2. Розробка меню, налаштувань, переходи між рівнями

Головне меню реалізовано як окрема сцена Unity з власною логікою відображення і реагування на події. Його архітектура побудована за принципом модульності: кожен розділ меню — це окремий елемент, пов'язаний із відповідним менеджером (наприклад, менеджером налаштувань або менеджером сцен).

Користувач із головного меню може:

- розпочати нову гру;
- обрати конкретний рівень (через систему розблокування);

- перейти до меню налаштувань;
- ознайомитися з правилами гри;
- вийти з гри.

Для цього реалізовано окремий *UI Layer*, побудований за допомогою компонентів *Canvas*, *Button*, *Text*, *Panel* тощо.

Функціонал меню налаштувань реалізовано через окремий розділ головного меню. Основна мета — дати гравцеві інструмент контролю над ігровим процесом.

Передбачені розділи:

- гучність;
- мову інтерфейсу;
- мапа рівнів;

Це дозволяє підтримувати індивідуальні конфігурації для користувача, що сприяє комфорту.

Переходи між рівнями

Один із елементів ігрового процесу — це забезпечення плавного і логічного переходу між рівнями. Для цього у проєкті реалізовано *Scene Management System*, яка відповідає за:

- завантаження наступного рівня після успішного проходження поточного;
- збереження інформації про досягнення гравця;
- реалізацію механізму розблокування наступного рівня після виконання умов (перемога, досягнення ворога);
- повернення до головного меню або повторне проходження рівня.

Переходи реалізуються через *SceneManager.LoadScene()* (див. рис.3.5) [\[45\]](#) з попередньо визначеною чергою рівнів за допомогою налаштувань у *Scene List*. Такий спосіб є самим зручним та доцільним з точки зору ефективності, адже він не прив'язується до назви і т.д., а записує сцену як такову, тому незалежно від

ваших подальших дій вона буде зберігати свою послідовність доки існуватиме сцена або ви самі його не зміните(див. рис.3.1-4). Знайти метод можна через $\Rightarrow$ File $\Rightarrow$ Build Profiles $\Rightarrow$ Scene List. *Приклад роботи*

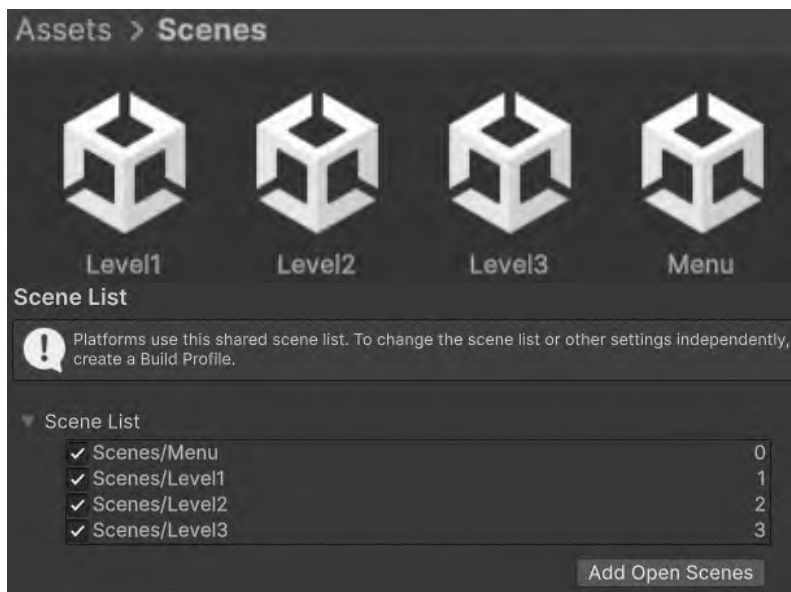


Рис. 3.1-2. Налаштування послідовності

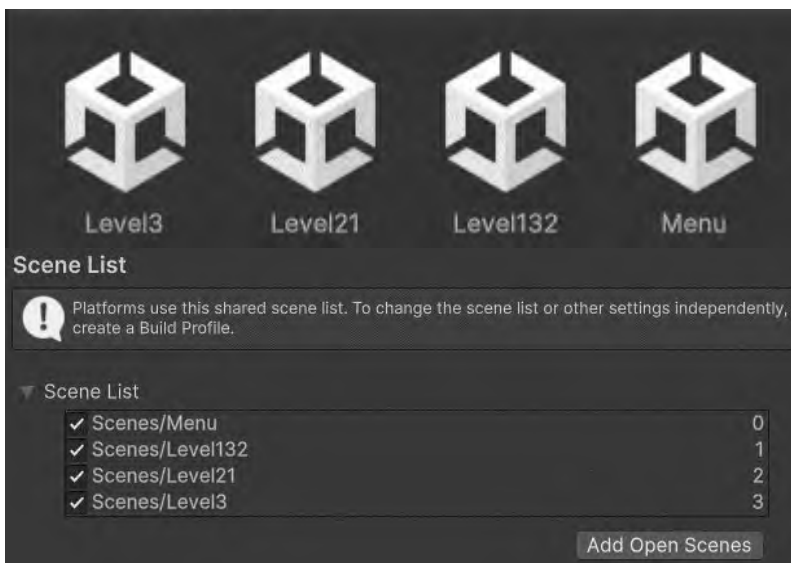


Рис.3.3-4. Після змін

```

1  using UnityEngine;
2  using UnityEngine.SceneManagement;
3  public class MainMenu : MonoBehaviour
4  {
5      public void PlayGame()
6      {
7          SceneManager.LoadScene(SceneManager.GetActiveScene().buildIndex + 1);
8      }
9  }

```

Рис. 3.5. Приклад коду для переходу між рівнями SceneManager.LoadScene()

Такий код зберігає має свою простоту, тому що наслідуює індекс поточної сцени +1 та посилає за цим індексом.

Для збереження цілісності ігрового процесу використано централізований *GameManager (BoardManager)*, що зберігає інформацію про поточний стан гри, номер рівня, та ключові події. Це дозволяє легко відновити гру при перериванні або після аварійного завершення програми.

Аналіз результатів тестування та ефективності

Тестування в розробці логічної гри відіграє ключову роль, бо має забезпечувати стабільність програмного коду та цілісність механік ігрової справедливості для відповідності розробленому продукту початковими педагогічними цілями. Інтелектуальна спрямованість проєкту, потребує прискіпливого тестування для перевірки ефективності структури головоломок та поведінки гравця у типових сценаріях.

Саме тестування було проведене за методологією *Qalight* [22]

У процесі тестування було застосовано комбінацію **модульного** та **ігрового** тестування. Модульне тестування охоплювало окремі компоненти гри — зокрема, системи руху фігур, обробки фізики, визначення переможних умов [10]. Для цього застосовувалися користувацькі тести на C# з перевіркою ключових умов, наприклад, допустимість ходу фігури або правильність виявлення зіткнення.

Найбільш важливим було ігрове тестування, яке включало безпосередню взаємодію з прототипом гри з боку незалежних учасників. Це дозволило з'ясувати складність задач, що пропонуються, та виявити неочевидні варіанти рішень.

Тестування логіки рішень припускає, що кожен рівень повинен містити одну або кілька логічно обґрунтованих стратегій досягнення мети, відповідно до правил переміщення шахових фігур та властивостей фізичного середовища. Рівні будувалися таким чином, щоб виключити можливість випадкового проходження або ігнорування головоломки.

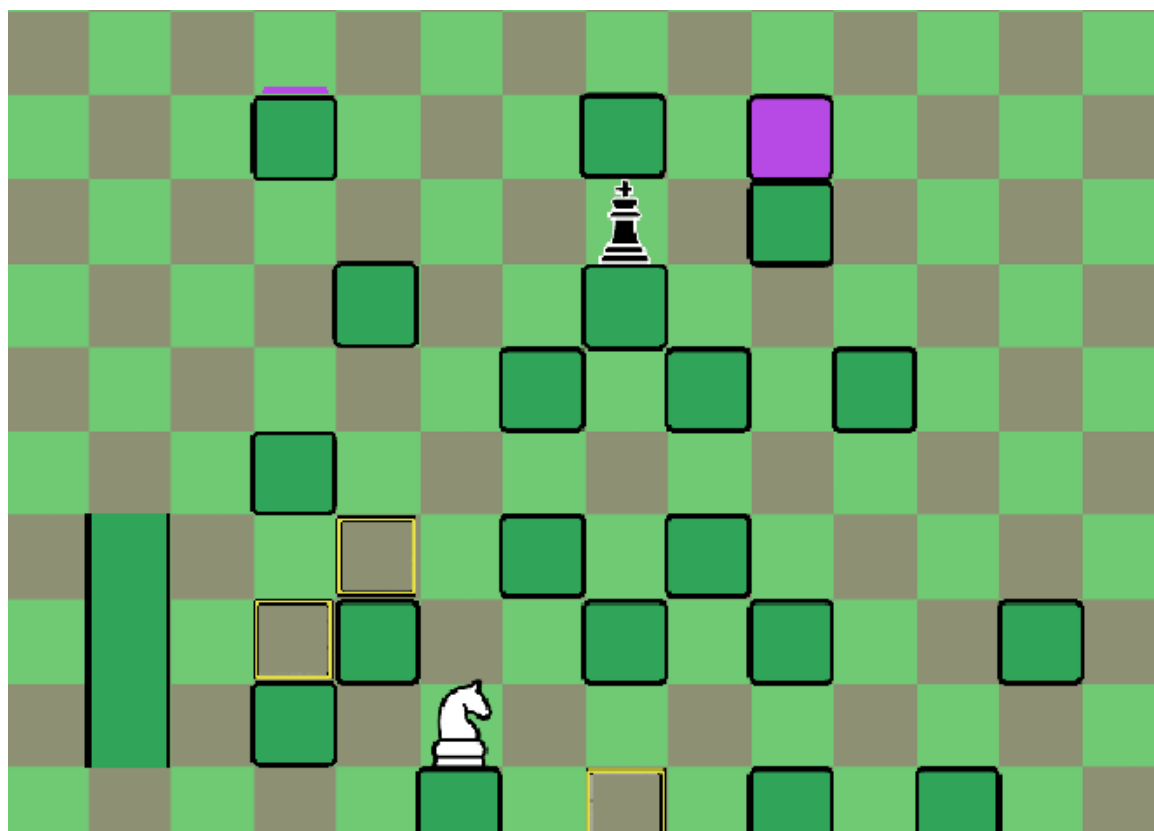


Рис.3.6. Бонусний рівень. Приклад

Для кожного рівня було складено карти рішень (див. рис.3.7), які описують допустимі послідовності ходів для досягнення ворожого короля. Вони стали основою для перевірки послідовності ходів, які справді ведуть до перемоги, та чи не існує альтернативних шляхів, які порушують задані правила.

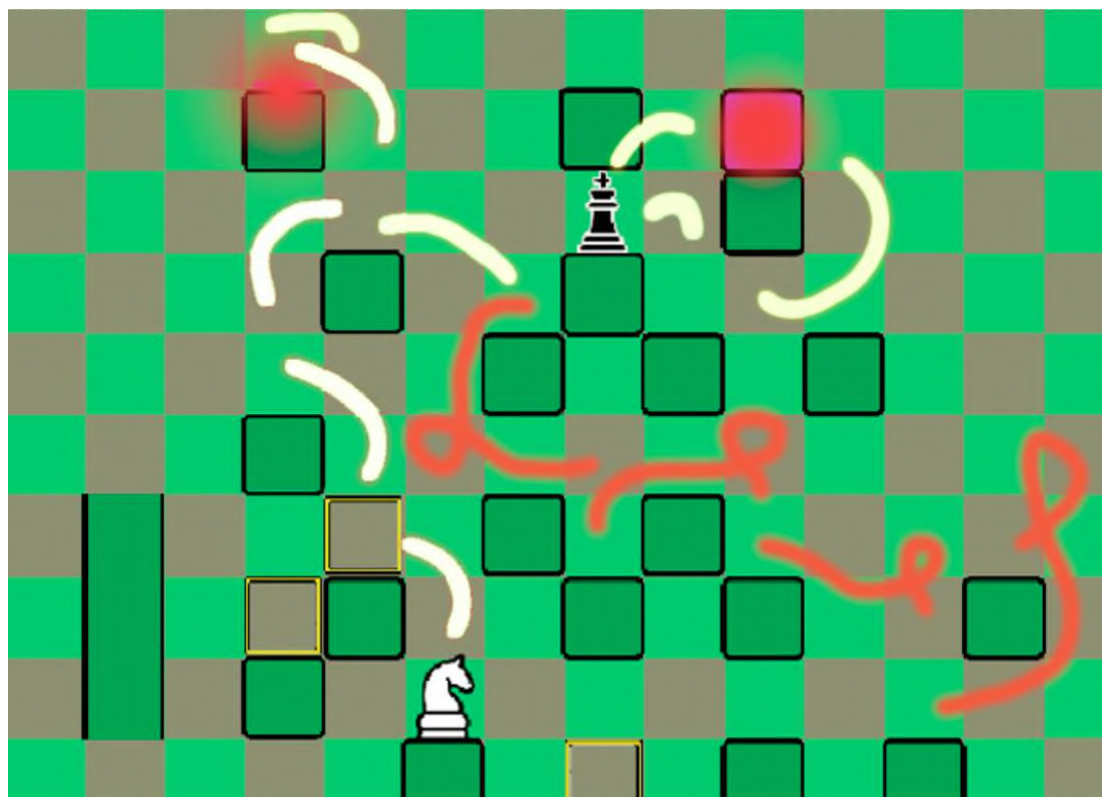


Рис. 3.7. Приклад вірного та хибного проходження

Важливою частиною перевірки стало тестування помилкових сценаріїв — наприклад, хід у забороненому напрямку, спроба перемістити ворожу фігуру, або ж переміщення фігури в умовах, коли вона має падати згідно з гравітацією. Усі ці сценарії були протестовані з метою забезпечення логічної реакції гри.

Аналіз результатів тестування

Тестування дозволило знайти низку критичних помилок на ранніх етапах розробки. Серед них — некоректна робота фізичного рушії під час зміни сцени, хибна маштабованість логіки ходів при зміні масштабу камери, а також надто велика складність деяких рівнів, що виходила за межі логічного мислення і перетворювалася на метод спроб і помилок. Після повторного тестування всі виявлені проблеми були усунуті, а головоломки приведені у відповідність до встановлених критеріїв:

- чіткість умов
- наявність єдиного розв'язку або обмеженої множини стратегій
- поступове зростання складності.

За тестуванням геймплею та логіки рішень. Етапи перевірки забезпечили контроль за кожним аспектом гри — від найпростіших рухів до заплутаних сценаріїв взаємодії, що дозволило досягти високого рівня цілісності.

3.3. Оцінка ефективності гри як логічного тренажера

Ефективність логічної гри як навчального інструмента базується на оцінці її здатності розвивати аналітичні навички, необхідні для вирішення нестандартних завдань. На відміну від розважального програмного продукту, логічний тренажер повинен забезпечувати приємне навантаження на мізки.

У процесі оцінки ефективності гри було враховано положення теорії розвитку інтелекту через ігрову діяльність. В основі підходу — теза про те, що мислення активізується в умовах вирішення завдань із багатьма умовами, але обмеженою кількістю рішень [25]. Саме така модель реалізована в розробленій грі: гравцеві пропонується обмежене ігрове поле з чіткими правилами, однак шлях до мети може мати кілька допустимих варіантів, що формують поле для логічного пошуку.

Для кількісного й якісного оцінювання ефективності були визначені такі ключові критерії:

Складність головоломок — поступове ускладнення рівнів дозволяє судити про ефективність гри: початкові рівні формують базове розуміння механіки, а наступні змушують користувача комбінувати логічні підходи.

Кількість помилкових спроб — відображає рівень логічного навантаження: при адекватній складності користувач повинен здійснювати помилки, однак мати змогу на їх основі знайти правильну стратегію.

Рівень повторного використання рівнів — свідчить про ігрову зацікавленість та мотивацію до самовдосконалення.

Усі ці критерії були реалізовані в системі внутрішньоїігрової статистики, яка зберігає дані про дії гравця в процесі проходження.

Результати первинного користувацького тестування

Пілотне тестування гри проводилося за участі групи людей (віком від 7 до 22 років). Їм було запропоновано пройти 5 рівнів гри, після чого було проведене опитування, що дозволила зібрати суб'єктивні оцінки ігрової логіки, складності та освітнього потенціалу.

Основні результати:

80% зазначили, що гра вимагає концентрації зокрема при нестандартних нових видах перешкод.

30% вказали на покращення здатності до планування дій наперед, особливо у складних рівнях, де фігури потрібно використовувати в суворій послідовності.

40% відзначили, що при повторному проходженні рівнів знаходять альтернативні розв'язки, що свідчить про відкритість простору рішень.

100% учасників змогли вивести правильну логіку роботи хоча б однієї фігури (наприклад, ефект падіння при пересуванні на край платформи), що говорить про легке засвоєння навчальних закономірностей на практиці.

Інтерпретація результатів

Отримані дані говорять про хороший потенціал гри як логічного тренажера, з використанням класичних шахових правил у новому фізичному вимірі. Поєднання стратегічних обмежень з елементами ризику по типу падіння чи нестійких платформи, стимулює гравця до більш гнучкого мислення та творчого підходу до пошуку рішення.

Крім того, постійна присутність єдиної забезпечує фокусовану діяльність гравця, не розсіюючи увагу зайвими механіками. Це відповідає меті, згідно з якою

ігрове середовище має бути навантаженим, але не перевантаженим другорядними елементами.

У результаті дослідження встановлено, що розроблена гра ефективно виконує функцію логічного тренажера. Гра може бути використана як допоміжний засіб для розвитку логіки у дітей старшого шкільного віку, студентів технічних спеціальностей, а також як інструмент відпочинку від фізичної активності.

Висновки до розділу 3

У третьому розділі кваліфікаційної роботи було розглянуто практичні аспекти реалізації логічної відеогри, побудованої на комбінації шахових правил та елементів фізичної симуляції. На основі завдань і технічного проектування були створені ключові компоненти, що забезпечують інтерактивність і навчальну цінність гри.

Було реалізовано інструмент редактора рівнів, що дозволяє створювати й модифікувати логічні задачі без втручання в кодову базу. Цей компонент є необхідною умовою масштабованості та подальшого розвитку гри як освітнього продукту.

Розроблено повноцінну **систему меню та переходів між рівнями** [\[18\]](#), що відповідає вимогам зручності користувача та забезпечує структуровану навігацію в ігровому середовищі.

Особливу увагу було приділено **тестуванню геймплею** з урахуванням логічної обґрунтованості задач. Це дозволило забезпечити правильну роботу механік, послідовність ускладнення рівнів та уникнення зайвої випадковості в досягненні мети. Відповідно, отримані результати стали основою для оцінки ефективності гри як логічного тренажера.

Проведений аналіз свідчить, що обране технічне рішення вдало поєднує класичні шахові алгоритми із засобами фізичної взаємодії об'єктів та інтерактивної фізики. Це створює тип логічної гри, здатної легко зайняти ваші

думки на 20 хвилин. Таким чином, поставлені цілі третього розділу були досягнуті, що закладає основу для підсумкових висновків кваліфікаційного дослідження.

ВИСНОВКИ

У результаті виконання кваліфікаційної роботи було досягнуто поставлену мету — здійснено повний цикл розробки інтерактивної логічної відеогри з елементами шахових механік і фізичної симуляції, що спрямована на розвиток аналітичного мислення користувача. На основі аналізу наукових джерел, огляду сучасних інструментів програмування, а також практичної реалізації проекту, можна сформулювати такі висновки:

Теоретичний аналіз логічних ігор показав, що їх вплив на когнітивні функції має високий потенціал серед ігрових середовищ як освітньо-розвивальний інструмент. Особлива увага була приділена використанню шахових алгоритмів як основі гри, що полегшує розуміння задачі, але змушує вмикати мізки, просторове мислення, передбачення варіантів і так далі.

Проведено огляд сучасного інструментарію для розробки ігор, серед якого було обрано використання платформою Unity у поєднанні з мовою програмування C#. Unity забезпечує високий рівень інтеграції з фізичними рушіями, модульність, мультиплатформенність, підтримку 2D-графіки та просте налагодження, що є критичними чинниками при створенні навчальної гри. Мова C#, у свою чергу, надала можливість реалізувати складні логічні сценарії, структури даних та перевірки коректності ігрових дій.

В аналітичній частині здійснено моделювання ігрового процесу, зокрема описано цілі гравця, логіку досягнення мети, рівні складності, правила руху фігур згідно з шаховою логікою та особливості фізичної взаємодії об'єктів. Розроблена архітектура гри виявилася придатною для масштабування та доповнення новими механіками, що забезпечує її потенціал подальшого розвитку.

Практична реалізація гри охоплювала створення шахових фігур, з урахуванням класичних алгоритмів їх переміщення, а також фізичних властивостей, таких як падіння, гравітація, зіткнення. Унікальність гри полягає в

синтезі стратегічної логіки шахів і динаміки фізичних взаємодій, що створює новий тип логічної задачі з підвищеним рівнем залученості користувача.

У процесі тестування було перевірено стабільність геймплею, логіку сценаріїв і зручність інтерфейсу. Аналіз взаємодії гравця з грою показав, що користувачі активно залучаються до процесу вирішення завдань, шукають альтернативні варіанти ходів і виявляють інтерес до повторного проходження рівнів — це підтверджує ефективність проєкту як логічного тренажера.

Отже, розроблена гра може бути використана як засіб розвитку логіки, просторового мислення та стратегії та база для подальших досліджень у сфері ігрової педагогіки. Крім того, вона демонструє приклад застосування програмування на C# та глибокого розуміння архітектури Unity в задачах освітнього характеру.

Таким чином, кваліфікаційна робота успішно продемонструвала можливість створення сучасного ігрового продукту, який поєднує освітню функцію з привабливістю геймплею, а також виявила потенціал подальшого розвитку як у технічному, так і в методичному напрямку.

СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ

1. Вказівки користувача. Які переваги використання механізму шаблонів?
URL: <https://kiyvagrolis.com.ua/yaki-perevagi-vikoristannya-mekhanizmu-shabloniv/> (дата звернення: 16.05.2025)
2. Вплив комп'ютерних ігор на розвиток логічного мислення URL:
https://kogpa.edu.ua/images/main_dir/add_disciples_zah_pidhot/vplyv_igor.pdf
(дата звернення: 12.05.2025)
3. Логічна Гра "Pawnbarian" (2021). Мехатіки, опис та рецензії
URL: <https://store.steampowered.com/app/1142080/Pawnbarian/?l=ukrainian> (дата звернення: 18.05.2025)
4. Деталі гри "5D Chess With Multiverse Time Travel" URL:
https://en.wikipedia.org/wiki/5D_Chess_with_Multiverse_Time_Travel (дата звернення: 11.05.2025)
5. Інтеграція ігрових елементів та цифрових інструментів у сучасні методи навчання програмуванню
URL: <https://journals.uran.ua/texel/article/view/316932/307437> (дата звернення: 15.05.2025)
6. Інтегроване середовище розробки. Визначення.
URL: <https://surl.li/wodgfx> (дата звернення: 16.05.2025)
7. Метод проб і помилок. Визначення.
URL: <https://surl.li/nbgzmm> (дата звернення: 16.05.2025)
8. Мова програмування C#. Навчальний посібник
URL: <https://ela.kpi.ua/server/api/core/bitstreams/6d82174e-d675-4d2c-b096-3dbe2ce4a534/content> (дата звернення: 16.05.2025)
9. Модель—представлення—контролер — архітектурний шаблон, який використовується під час проєктування та розробки програмного забезпечення.
URL: <https://surli.cc/vqjhdc> (дата звернення: 17.05.2025)
10. Модульне тестування в програмуванні. Визначення.
URL: <https://surl.lu/xhpcfm> (дата звернення: 11.05.2025)
11. Навчальний модуль «Розробка комп'ютерних ігор за допомогою Unity 3D»
URL: https://ec.europa.eu/programmes/erasmus-plus/project-result-content/7c7d9761-4402-467c-adab-5b09579cb8fd/2017_KNTU_Module2.pdf (дата звернення: 12.05.2025)
12. Настільна логічна гра для інтелектуалів – Шахи. Дослідження.
URL: <https://www.udau.edu.ua/ua/other-news/shaxi-gra-dlya-intelektualiv.html>
(дата звернення: 03.05.2025)
13. Основні кроки створення відеогри
URL: <https://surl.lu/knbwju> (дата звернення: 15.05.2025)
14. Початок занять шахами у дошкільному віці – центр розвитку “Діалог”
URL: <https://centr-dialog.com.ua/chess> (дата звернення: 08.05.2025)

15. Правила шахів. Вікіпедія
URL: <https://surl.li/rlzgdy> (дата звернення: 06.05.2025)
16. Предметно-орієнтоване програмування. Розмежування повноважень URL:
<https://surl.li/hboqwi> (дата звернення: 08.05.2025)
17. Психологічні ПОЗИТИВНІ ЕФЕКТИ ВПЛИВУ ВІДЕОІГОР
URL: <https://ekmair.ukma.edu.ua/server/api/core/bitstreams/52a0901b-77f5-4081-a16d-8d49b03591c7/content> (дата звернення: 08.05.2025)
18. Що таке “Редактор рівнів”. Навіщо він потрібен, та як користуватись
URL: https://uk.wikipedia.org/wiki/Редактор_рівнів (дата звернення: 08.05.2025)
19. Рейтинг мов програмування в Україні та світі 2025
URL: <https://justsmart.com.ua/uk/blog/rejtyng-mov-programuvannya-v-ukrayini-ta-sviti-2025/> (дата звернення: 16.05.2025)
20. Рейтинг мов програмування 2025. TypeScript і Python — найпопулярніші, частки C# та Java зменшуються
URL: <https://dou.ua/lenta/articles/language-rating-2025/> (дата звернення: 05.05.2025)
21. Розвиток логічного мислення у дітей старшого дошкільного віку
URL: https://journals.maup.com.ua/index.php/pedagogy/article/download/4366/4680/5061?utm_source=chatgpt.com (дата звернення: 16.05.2025)
22. Тестування веб-проектів: основні етапи та поради
URL: <https://qalight.ua/baza-znaniy/testuvannya-veb-proektiv-osnovni-etapi-ta-poradi/> (дата звернення: 05.05.2025)
23. Що робить фізичний рушій. Методи використання.
URL: https://uk.wikipedia.org/wiki/%D0%A4%D1%96%D0%B7%D0%B8%D1%87%D0%BD%D0%B8%D0%B9_%D1%80%D1%83%D1%88%D1%96%D0%B9 (дата звернення: 16.05.2025)
24. Фізичний рушій. Симуляцію фізичних законів реального світу
URL: <https://surli.cc/cxqinx> (дата звернення: 16.05.2025)
25. Функціонування творчого мислення в умовах інформаційної невизначеності
URL: https://lib.iitta.gov.ua/id/eprint/742710/1/%D0%9C%D0%9E%D0%9D%D0%9E%D0%93%D0%A0%D0%90%D0%A4%D0%86%D0%AF_%D0%9C%D0%BE%D0%BB%D1%8F%D0%BA%D0%BE_2024_.pdf?utm_source (дата звернення: 14.05.2025)
26. Характеристика логічних ігор. Комп'ютерні логічні ігри
URL: <https://studfile.net/preview/7363365/page:6/> (дата звернення: 14.05.2025)
25. Глибокий опис C (мова програмування), в чому переваги.
URL: <https://surli.cc/gcuild> (дата звернення: 14.05.2025)
26. Konovalenko I. .NET-C# Приклади рішень проблем початківців
URL: <https://elartu.tntu.edu.ua/bitstream/lib/32825/1/Konovalenko%20I.%20.NET-C%23.pdf> (дата звернення: 02.05.2025)

27. Microsoft Word - C#_last. Посібник по мові C#
URL: https://ami.lnu.edu.ua/wp-content/uploads/2017/05/C_sharp.pdf (дата звернення: 02.05.2025)
28. Unity (ігровий рушій). Можливості, історія, характеристики.
URL: <https://surl.lu/zwearq> (дата звернення: 28.04.2025)
29. Unity (store): 5D Chess With Multiverse Time Travel
URL: https://en.wikipedia.org/wiki/5D_Chess_with_Multiverse_Time_Travel (дата звернення: 08.05.2025)
30. JSON, метод завантаження ігрових рівнів в спрощеному методі
URL: <https://uk.wikipedia.org/wiki/JSON> (дата звернення: 16.05.2025)
31. Lichess, гра на шахи онлайн
URL: <https://uk.wikipedia.org/wiki/Lichess> (дата звернення: 16.05.2025)
32. Into the Breach, гра по шаховим принципам, з іншою графікою.
URL: https://uk.wikipedia.org/wiki/Into_the_Breach (дата звернення: 03.05.2025)
33. Відеогра-головоломка. Визначення.
URL: https://en.wikipedia.org/wiki/Puzzle_video_game (дата звернення: 16.05.2025)
34. 71% of the biggest mobile games use a Unity SDK
URL: https://www.pocketgamer.biz/71-of-the-biggest-mobile-games-use-a-unity-sdk/?utm_source=chatgpt.com (дата звернення: 16.05.2025)
35. Changing BoxCollider2D size to match sprite bounds at runtime
URL: <https://discussions.unity.com/t/changing-boxcollider2d-size-to-match-sprite-bounds-at-runtime/550242> (дата звернення: 16.05.2025)
36. Discussions.unity: How do you set up a GameManager?
URL: <https://discussions.unity.com/t/help-how-do-you-set-up-a-gamanager/473071> (дата звернення: 18.05.2025)
37. Fransson H. AgentChess: An Agent Chess Approach (thesis, 2003)
URL: <http://urn.kb.se/resolve?urn=urn:nbn:se:bth-5508> (дата звернення: 09.05.2025)
38. GitHub, міжнародна платформа програмістів
URL: <https://github.com/> (дата звернення: 01.05.2025)
39. How do games develop our critical thinking skills?
URL: <https://arc.academy/news/how-do-games-develop-our-critical-thinking-skills/> (дата звернення: 16.05.2025)
40. Mataruga D. Robustness of NLUI: A Chess implementation (thesis, 2016)
URL: <http://urn.kb.se/resolve?urn=urn:nbn:se:kth:diva-186511> (дата звернення: 16.05.2025)
41. Physics2D.gravity. Description Acceleration due to gravity.

- URL: <https://docs.unity3d.com/6000.1/Documentation/ScriptReference/Physics2D-gravity.html>
42. Physics solutions for game development
URL: <https://unity.com/solutions/programming-physics> (дата звернення: 16.05.2025)
43. Portal, гра на логічне мислення за використанням фізики.
URL: <https://uk.wikipedia.org/wiki/Portal> (дата звернення: 16.05.2025)
44. Robustness of NLUIs A Chess Implementation DANI MATARUGA
URL: <https://kth.diva-portal.org/smash/get/diva2:927500/FULLTEXT01.pdf> (дата звернення: 07.05.2025)
45. Scene Manager. LoadScene. Пояснення налаштування черни появи сцен.
URL: <https://docs.unity3d.com/6000.1/Documentation/ScriptReference/SceneManager.LoadScene.LoadScene.html> (дата звернення: 07.05.2025)
46. Tech Crunch. Інтерв'ю з власником Unity
URL: https://techcrunch.com/2018/09/05/unity-ceo-says-half-of-all-games-are-built-on-unity/?utm_source
47. The role of video games in enhancing managers' strategic thinking and cognitive abilities: An experiential survey
URL: <https://www.sciencedirect.com/science/article/abs/pii/S1875952124000624> (дата звернення: 04.05.2025)
48. Yang Y. Wu: A Cultural Export Game with Dynamic Difficulty. 2020.
URL: <https://digitalcommons.wpi.edu/etd-theses/1361> (дата звернення: 16.05.2025)
49. Yasufuku K., Katou G., Shoman S. Game Engine (Unity, Unreal Engine). The Journal of ITE, 2017.
URL: <https://doi.org/10.3169/itej.71.353> (дата звернення: 14.05.2025)

ДОДАТКИ

Додаток А

Побудова логіки фігур мовою С#

Bishop

Cusing UnityEngine;

```
public class Bishop : Piece
{
    private static readonly Vector2Int[] directions = {
        new Vector2Int(1, 1), new Vector2Int(-1, 1),
        new Vector2Int(-1, -1), new Vector2Int(1, -1)
    };

    public override void ShowMoves()
    {
        foreach (var dir in directions)
        {
            int newX = x + dir.x;
            int newY = y + dir.y;

            while (board.IsInsideBoard(newX, newY))
            {
                Tile tile = board.GetTileAt(newX, newY);
                if (tile.currentPiece == null)
                {
                    board.SpawnMovePlate(this, newX, newY);
                }
                else
                {
                    if (tile.currentPiece.isWhite != isWhite)
                    {
                        board.SpawnMovePlate(this, newX, newY);
                        break;
                    }
                }

                newX += dir.x;
                newY += dir.y;
            }
        }
    }
}
```

King

```

using UnityEngine;

public class King : Piece
{
    public override void ShowMoves()
    {
        for (int dx = -1; dx <= 1; dx++)
        {
            for (int dy = -1; dy <= 1; dy++)
            {
                if (dx == 0 && dy == 0) continue;

                int newX = x + dx, newY = y + dy;

                Tile tile = board.GetTileAt(newX, newY);
                if (tile != null && (tile.currentPiece == null || tile.currentPiece.isWhite != isWhite))
                    board.SpawnMovePlate(this, newX, newY);
            }
        }
    }
}

```

Rook

```

using UnityEngine;

public class Rook : Piece
{
    public override void ShowMoves()
    {
        LinearMoves(1, 0);
        LinearMoves(-1, 0);
        LinearMoves(0, 1);
        LinearMoves(0, -1);
    }

    void LinearMoves(int dx, int dy)
    {
        int cx = x + dx, cy = y + dy;
        while (board.IsInsideBoard(cx, cy))
        {
            Tile tile = board.GetTileAt(cx, cy);
            if (tile.currentPiece == null)
                board.SpawnMovePlate(this, cx, cy);
        }
    }
}

```

```

else
{
    if (tile.currentPiece.isWhite != isWhite)
        board.SpawnMovePlate(this, cx, cy);
    break;
}
cx += dx;
cy += dy;
}
}
}

```

Queen

```

using UnityEngine;

public class Queen : Piece
{
    public override void ShowMoves()
    {
        LinearMoves(1, 0); LinearMoves(-1, 0);
        LinearMoves(0, 1); LinearMoves(0, -1);
        LinearMoves(1, 1); LinearMoves(-1, -1);
        LinearMoves(1, -1); LinearMoves(-1, 1);
    }

    void LinearMoves(int dx, int dy)
    {
        int cx = x + dx, cy = y + dy;
        while (board.IsInsideBoard(cx, cy))
        {
            Tile tile = board.GetTileAt(cx, cy);
            if (tile.currentPiece == null)
                board.SpawnMovePlate(this, cx, cy);
            else
            {
                if (tile.currentPiece.isWhite != isWhite)
                    board.SpawnMovePlate(this, cx, cy);
                break;
            }
            cx += dx;
            cy += dy;
        }
    }
}

```

Pawn

```

using UnityEngine;

public class Pawn : Piece
{
    public override void ShowMoves()
    {
        int direction = isWhite ? 1 : -1;
        int startRow = isWhite ? 1 : 6;

        // Прямий хід
        if (board.GetTileAt(x, y + direction)?.currentPiece == null)
        {
            board.SpawnMovePlate(this, x, y + direction);

            if (y == startRow && board.GetTileAt(x, y + 2 * direction)?.currentPiece == null)
                board.SpawnMovePlate(this, x, y + 2 * direction);
        }

        // Атака
        TryAttack(x - 1, y + direction);
        TryAttack(x + 1, y + direction);
    }

    void TryAttack(int targetX, int targetY)
    {
        Tile tile = board.GetTileAt(targetX, targetY);
        if (tile != null && tile.currentPiece != null && tile.currentPiece.isWhite != isWhite)
            board.SpawnMovePlate(this, targetX, targetY);
    }
}

```

Knight

```

using UnityEngine;

public class Knight : Piece
{
    public override void ShowMoves()
    {
        Vector2Int[] moves = {
            new Vector2Int(2, 1), new Vector2Int(1, 2),
            new Vector2Int(-1, 2), new Vector2Int(-2, 1),
            new Vector2Int(-2, -1), new Vector2Int(-1, -2),
            new Vector2Int(1, -2), new Vector2Int(2, -1)
        }
    }
}

```

```
};  
foreach (Vector2Int move in moves)  
{  
    int newX = x + move.x;  
    int newY = y + move.y;  
  
    Tile tile = board.GetTileAt(newX, newY);  
    if (tile != null && (tile.currentPiece == null || tile.currentPiece.isWhite != isWhite))  
    {  
        board.SpawnMovePlate(this, newX, newY);  
    }  
}  
}
```

Налаштування та організація ігрового середовища

GameManager (BoardManager)

```
using UnityEngine;

using System.Collections.Generic;

// Клас BoardManager відповідає за створення шахівниці, розміщення фігур і керування їхньою взаємодією.
public class BoardManager : MonoBehaviour
{
    [Header("Prefabs")]
    // Префаб для плитки шахівниці
    public GameObject tilePrefab;
    // Префаб для індикатора можливого ходу (MovePlate)
    public GameObject movePlatePrefab;
    // Префаби для білих шахових фігур
    public GameObject Knight;
    public GameObject pawnPrefab;
    public GameObject kingPrefab;
    public GameObject queenPrefab;
    public GameObject bishopPrefab;
    public GameObject rookPrefab;
    // Префаби для чорних шахових фігур
    public GameObject knightPrefabBlack;
    public GameObject pawnPrefabBlack;
    public GameObject kingPrefabBlack;
    public GameObject queenPrefabBlack;
    public GameObject bishopPrefabBlack;
    public GameObject rookPrefabBlack;

    [Header("Board Size")]
    // Розміри шахівниці (за замовчуванням 8x8)
    public int boardWidth = 8;
    public int boardHeight = 8;

    // Двовимірний масив для зберігання плиток шахівниці
    private Tile[,] tiles;
    // Список для зберігання всіх MovePlates
    private List<GameObject> movePlates = new List<GameObject>();
    // Списки для відстеження активних фігур
    private List<Piece> whitePieces = new List<Piece>();
```

Продовження додатка Б

```

private List<Piece> blackPieces = new List<Piece>();
// Посилання на чорного короля
private Piece blackKing;

// Метод для отримання плитки за координатами (x, y)
public Tile GetTileAt(int x, int y)
{
    if (IsInsideBoard(x, y))
        return tiles[x, y];
    return null;
}

// Метод, який викликається при ініціалізації об'єкта
void Start()
{
    tiles = new Tile[boardWidth, boardHeight];
    CreateBoard();
    SpawnPiece(Knight, 8, 3, true);
    SpawnPiece(kingPrefab, 10, 10, false);
}

// Метод для створення шахівниці
void CreateBoard()
{
    for (int x = 0; x < boardWidth; x++)
    {
        for (int y = 0; y < boardHeight; y++)
        {
            GameObject obj = Instantiate(tilePrefab);
            obj.transform.position = new Vector3(x, y, 0);
            Tile tile = obj.GetComponent<Tile>();
            if (tile == null)
                tile = obj.AddComponent<Tile>();
            tile.SetCoordinates(x, y);
            tiles[x, y] = tile;
            SpriteRenderer sr = obj.GetComponent<SpriteRenderer>();
            if ((x + y) % 2 == 0)
                sr.color = new Color32(141, 144, 115, 255);
            else
                sr.color = new Color32(112, 202, 116, 255);
        }
    }
}

```

Продовження додатка Б

```

// Метод для розміщення шахової фігури на дошці
public void SpawnPiece(GameObject prefab, int x, int y, bool isWhite)
{
    if (!isWhite)
    {
        if (prefab == Knight) prefab = knightPrefabBlack;
        else if (prefab == bishopPrefab) prefab = bishopPrefabBlack;
        else if (prefab == pawnPrefab) prefab = pawnPrefabBlack;
        else if (prefab == rookPrefab) prefab = rookPrefabBlack;
        else if (prefab == queenPrefab) prefab = queenPrefabBlack;
        else if (prefab == kingPrefab) prefab = kingPrefabBlack;
    }
    GameObject obj = Instantiate(prefab);
    obj.transform.position = new Vector3(x, y, -1);
    Piece piece = obj.GetComponent<Piece>();
    if (piece == null)
        piece = obj.AddComponent<Piece>();
    piece.board = this;
    piece.x = x;
    piece.y = y;
    piece.isWhite = isWhite;
    piece.UpdateSprite();
    if (isWhite)
        whitePieces.Add(piece);
    else
    {
        blackPieces.Add(piece);
        if (prefab == kingPrefabBlack)
            blackKing = piece;
    }
    bool hasSupport = IsInsideBoard(x, y - 1) && GetTileAt(x, y - 1).currentPiece != null;
    if (hasSupport)
        tiles[x, y].currentPiece = piece;
    else
    {
        piece.transform.position = new Vector3(x, y - 0.25f, -1);
        piece.EnablePhysics();
    }
}

// Метод для створення індикатора можливого ходу (MovePlate)
public void SpawnMovePlate(Piece piece, int x, int y)
{
    GameObject mp = Instantiate(movePlatePrefab);

```

```

mp.tag = "MovePlate";
mp.transform.position = new Vector3(x, y, -2);
movePlates.Add(mp);
MovePlate plate = mp.GetComponent<MovePlate>();
if (plate == null)
    plate = mp.AddComponent<MovePlate>();
plate.targetX = x;
plate.targetY = y;
plate.piece = piece;
plate.board = this;
}

// Метод для переміщення фігури
public void MovePiece(Piece piece, int targetX, int targetY)
{
    Tile oldTile = GetTileAt(piece.x, piece.y);
    if (oldTile != null) oldTile.currentPiece = null;
    piece.x = targetX;
    piece.y = targetY;
    bool noSupport = !IsInsideBoard(targetX, targetY - 1)
        || (GetTileAt(targetX, targetY - 1) != null && GetTileAt(targetX, targetY -
1).currentPiece == null);
    if (noSupport)
    {
        piece.transform.position = new Vector3(targetX, targetY, -1);
        piece.EnablePhysics();
    }
    else
    {
        piece.transform.position = new Vector3(targetX, targetY, -1);
        Tile newTile = GetTileAt(targetX, targetY);
        if (newTile != null)
        {
            if (newTile.currentPiece != null && newTile.currentPiece.isWhite == piece.isWhite)
                piece.EnablePhysics();
            else
            {
                newTile.currentPiece = piece;
                piece.isFalling = false;
                piece.landed = true;
            }
        }
    }
}

CheckGameEnd();

```

```

    ClearMovePlates();
}

// Метод для перевірки завершення гри
private void CheckGameEnd()
{
    if (blackKing == null || !blackPieces.Contains(blackKing))
        EndGame("Перемога білих! Чорний король знищений.");
    if (whitePieces.Count == 0)
        EndGame("Перемога чорних! Білі фігури знищені.");
}

// Метод для завершення гри
private void EndGame(string message)
{
    foreach (Piece piece in whitePieces)
        if (piece != null) piece.enabled = false;
    foreach (Piece piece in blackPieces)
        if (piece != null) piece.enabled = false;
    UnityEngine.Debug.Log(message);
}

// Метод для видалення всіх MovePlates
public void ClearMovePlates()
{
    foreach (var plate in movePlates)
        if (plate != null) Destroy(plate);
    movePlates.Clear();
}

// Метод для видалення фігури зі списків (викликається з Piece при знищенні)
public void RemovePiece(Piece piece)
{
    if (piece.isWhite)
        whitePieces.Remove(piece);
    else
        blackPieces.Remove(piece);
}

// Перевіряє, чи координати (x, y) знаходяться в межах шахівниці
public bool IsInsideBoard(int x, int y)
{
    return x >= 0 && y >= 0 && x < boardWidth && y < boardHeight;
}
}

```

Moveplate

```

using UnityEngine;

// Клас можливого ходу для фігури
public class MovePlate : MonoBehaviour
{
    public int targetX, targetY;
    public Piece piece;
    public BoardManager board;

    void OnMouseDown()
    {
        Collider2D[] colliders = Physics2D.OverlapPointAll(new Vector2(targetX, targetY));
        foreach (var col in colliders)
        {
            if (col.gameObject.CompareTag("Ground"))
                return;
        }
        Tile targetTile = board.GetTileAt(targetX, targetY);
        if (targetTile == null) return;
        if (targetTile.currentPiece != null)
        {
            Piece targetPiece = targetTile.currentPiece;
            if (targetPiece.isWhite != piece.isWhite)
            {
                board.RemovePiece(targetPiece);
                Destroy(targetPiece.gameObject);
            }
            else
                return;
        }
        board.MovePiece(piece, targetX, targetY);
    }
}

```

Piece

```

using UnityEngine;

// Клас Piece представляє шахову фігуру
public class Piece : MonoBehaviour
{
    public int x, y;
}

```

Продовження додатка Б

```

public bool isWhite;
public BoardManager board;

public Sprite whiteSprite;
public Sprite blackSprite;

private Rigidbody2D rb;
public bool isFalling = false;
public bool landed = false;

void Awake()
{
    rb = GetComponent<Rigidbody2D>();
    if (rb == null)
        rb = gameObject.AddComponent<Rigidbody2D>();
    rb.bodyType = RigidbodyType2D.Static;
    rb.gravityScale = 1f;
    rb.constraints = RigidbodyConstraints2D.FreezeRotation;
    var col = GetComponent<BoxCollider2D>();
    if (col == null)
        gameObject.AddComponent<BoxCollider2D>();
}

public void OnMouseDown()
{
    if (isFalling) return;
    board.ClearMovePlates();
    ShowMoves();
}

public virtual void ShowMoves() { }

public void UpdateSprite()
{
    var sr = GetComponent<SpriteRenderer>();
    if (sr) sr.sprite = isWhite ? whiteSprite : blackSprite;
}

public void EnablePhysics()
{
    if (rb == null)
        rb = GetComponent<Rigidbody2D>();
    rb.bodyType = RigidbodyType2D.Dynamic;
    rb.gravityScale = 1f;
}

```

```

        rb.constraints = RigidbodyConstraints2D.FreezePositionX |
RigidbodyConstraints2D.FreezeRotation;
        isFalling = true;
        landed = false;
    }

    public void OnCollisionStay2D(Collision2D col)
    {
        if (!isFalling || landed) return;
        if (col.gameObject.CompareTag("Piece") || col.gameObject.CompareTag("Ground"))
        {
            Vector2 normal = col.GetContact(0).normal;
            if (normal.y > 0.1f)
            {
                int newX = Mathf.RoundToInt(transform.position.x);
                int newY = Mathf.RoundToInt(transform.position.y);
                transform.position = new Vector3(newX, newY, -1);
                x = newX;
                y = newY;
                if (rb) rb.bodyType = RigidbodyType2D.Static;
                Tile tile = board.GetTileAt(x, y);
                if (tile != null && tile.currentPiece == null)
                    tile.currentPiece = this;
                landed = true;
                isFalling = false;
            }
        }
    }

    public void OnTriggerEnter2D(Collider2D col)
    {
        if (!isFalling || landed) return;
        if (col.gameObject.CompareTag("Tile"))
        {
            int newX = Mathf.RoundToInt(transform.position.x);
            int newY = Mathf.RoundToInt(transform.position.y);
            bool hasSupport = board.IsInsideBoard(newX, newY - 1) && board.GetTileAt(newX, newY -
1).currentPiece != null;
            if (hasSupport)
            {
                rb.bodyType = RigidbodyType2D.Static;
                x = newX;
                y = newY;
                transform.position = new Vector3(x, y, -1);
            }
        }
    }

```

```

    Tile tile = board.GetTileAt(x, y);
    if (tile != null && tile.currentPiece == null)
        tile.currentPiece = this;
    landed = true;
    isFalling = false;
}
else if (newY <= 0)
{
    rb.bodyType = RigidbodyType2D.Static;
    x = newX;
    y = 0;
    transform.position = new Vector3(x, y, -1);
    Tile tile = board.GetTileAt(x, y);
    if (tile != null && tile.currentPiece == null)
        tile.currentPiece = this;
    landed = true;
    isFalling = false;
}
else
    transform.position = new Vector3(newX, newY - 0.05f, -1);
}
}
}

```

Tile

using UnityEngine;

// Клас Tile представляє клітинку шахівниці

public class Tile : MonoBehaviour

```

{
    public int x, y;
    public Piece currentPiece;

    public void SetCoordinates(int x, int y)
    {
        this.x = x;
        this.y = y;
    }
}

```